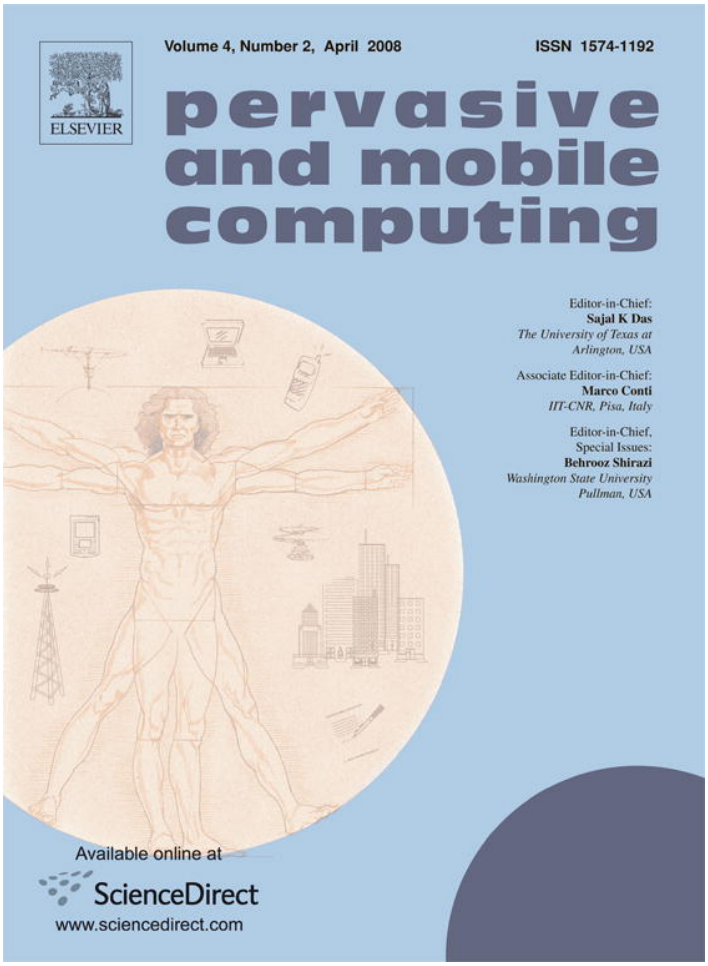
Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.



This article was published in an Elsevier journal. The attached copy is furnished to the author for non-commercial research and education use, including for instruction at the author's institution, sharing with colleagues and providing to institution administration.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright

Author's personal copy

Available online at www.sciencedirect.com
ScienceDirect
Pervasive and Mobile Computing 4 (2008) 182–215
pervasive and mobile computing
www.elsevier.com/locate/pmc

The ECORA framework: A hybrid architecture for context-oriented pervasive computing

Amir Padovitz[a,*], Seng W. Loke[b], Arkady Zaslavsky[a]

[a] *Monash Centre for Distributed Systems and Software Engineering, 900 Dandenong Road, Caulfield-East, Victoria, Australia*
[b] *Department of Computer Science and Computer Engineering Bundoora, Victoria, Australia*

Received 20 September 2006; received in revised form 28 September 2007; accepted 6 October 2007
Available online 26 November 2007

Abstract

An infrastructure approach to support context-aware pervasive computing is advantageous for rapid prototyping of context-aware distributed applications and beneficial for unifying modelling of context and reasoning in uncertain conditions. This paper presents the ECORA framework for context-aware computing, which is designed with a focus on reasoning about context under uncertainty and addressing issues of heterogeneity, scalability, communication and usability. The framework follows an agent-oriented hybrid approach, combining centralized reasoning services with context-aware, reasoning capable mobile software agents. The use of a centralized reasoning engine provides powerful reasoning capabilities and deploying context-aware mobile agents enables agility and robustness of components in the pervasive system. The design and implementation of the framework at different levels, as well as three case studies, are presented.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Context-aware pervasive computing; Context middleware; Mobile agents

* Corresponding author.
E-mail addresses: amirp@csse.monash.edu.au, amirp@microsoft.com (A. Padovitz), sloke@latrobe.edu.au (S.W. Loke), arkady.zaslavsky@csse.monash.edu.au (A. Zaslavsky).

1574-1192/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.pmcj.2007.10.002

Author's personal copy

1. Introduction

Recent advances in emerging pervasive computing technologies and communications have evolved into ample pioneering initiatives, leading towards a world in which computing systems are distributed, mobile, intelligent and cooperative. Central to the notion of pervasive systems is the ability to become context-aware. Context-aware computing endeavours to make systems aware of specific, valuable circumstances in the computing environment, and enable them to modify their behaviour accordingly. In the context of pervasive environments, context-aware systems are those which can respond (possibly intelligently) to context information acquired by any type of sensor (either physical or virtual). This, in turn, enhances services provided to users (including service personalization), makes pervasive systems intelligent by reacting (and possibly pro-acting) to changing circumstances, and promotes adaptability and autonomy of systems, liberating users from avoidable interactions.

From stand-alone applications, which demonstrate the benefits of using context, research has begun to look at modelling and reasoning about context in uncertain, distributed, open and heterogeneous computing environments. The inherent pervasiveness, heterogeneity and information uncertainty in such systems raises challenges for computing methods and feasible architectures that would support diverse clients and applications.

This paper presents ECORA (Extensible Context Oriented Reasoning Architecture) — a prototype framework for building context-aware applications, which are designed with a focus on reasoning about context under uncertainty and addressing issues of heterogeneity, scalability, communication and usability. Unique to the framework is an agent-oriented hybrid approach, combining centralized reasoning services with context-aware, reasoning capable mobile software agents. The use of a centralized reasoning engine provides powerful reasoning capabilities and (optionally) deploying context-aware mobile agents enables agility and robustness of components in the pervasive system. We assert that future computing pervasive systems would benefit from unified and effective reasoning capabilities and that mobility of context-aware software entities would enable realization of particularly challenging scenarios.

The framework is founded on novel theoretical underpinnings, enabling unifying context modelling and reasoning under uncertainty [28–30]. A context algebra [31] is developed as part of the framework, enriching the modelling and reasoning capabilities of components, and enabling agents to share different perspectives about situations and handle partial available sensory information. We also propose, develop and evaluate an effective communication mechanism between mobile agents of the framework, facilitating collaboration and adaptation of context-aware mobile agents in the pervasive system. Our proposed communication approach solves an existing problem of mobile agent communication, and enables new forms of messaging between mobile computing entities such as messaging based on the context of mobile entities.

The framework thus integrates three core capabilities, making the following contributions to building context infrastructure: (1) unified modelling and reasoning about context under uncertainty as a main ingredient in a context framework, (2) the ability of components to move physically and/or virtually within the network, and (3) implicit event-based communication between mobile and static entities within the framework.

Author's personal copy

In Section 2, we discuss related work and background, including current architectures of middleware for context-aware computing, and existing approaches to modelling and reasoning about context. In Sections 3 and 4, we present the architecture of the ECORA framework, including the reasoning engine and context-aware mobile agents. Section 3 presents the core services of the framework, and discusses issues of scalability and usability of the framework. Section 4 discusses the mobility and agility of the framework via using context-aware mobile agents and event-driven communication. In Section 5, we present case studies demonstrating different aspects of the framework, including rapid prototyping of context-aware applications, reasoning about situations via the framework, under uncertainty, and deploying context-aware mobile agents in the pervasive environment. We discuss and conclude the paper in Section 6.

2. Background

Much research in pervasive computing aims towards operating successfully in open, heterogeneous and context-aware computing environments [14,34,39]. Realizing this vision requires suitable approaches in modelling, reasoning and architecting effective context-aware systems that can handle reasoning in uncertain and rapidly changing environments. In recent years, research efforts have focused on these aspects of context, including context middleware and toolkits [6,9,5] for information acquisition, ontologies [43] that provide vocabularies to describe and share context information [7], different approaches to reason about context [12,23,36], and a variety of context models [45].

Of particular interest in dealing with complex and open pervasive systems are: (1) context models that represent context in a general way, including reasoning algorithms that can handle varying degrees of uncertainty, and (2) architectures that promote agility and autonomy of individual computing entities.

2.1. Agent-based middleware for pervasive computing

The agent paradigm has been adopted as a suitable way to architect middleware and applications for pervasive computing [6,17,18]. With agent-oriented computing, autonomous entities in the system individually manage specific tasks and cooperate among themselves with standardized protocols to achieve a greater goal. Agent-based middleware for ubiquitous computing environments is proposed in [36–38]. The architectural approach builds on the notion of agents as autonomous entities, which can be applications, services and devices in the ubiquitous environment. ACAI [17] is another agent-oriented infrastructure for context-aware computing. The design employs agents, which handle various issues in the lifecycle of a context-aware computing system. The overall architectural approach appears to be geared towards a centralized architecture with agents using the system's centralized services. The Context Broker architecture (CoBrA) [6,9] is another agent-oriented approach for building middleware for context-aware computing. While it mainly deals with a specific use case of smart rooms, the architectural approach follows the design of broker agents, which maintain a shared model of context. The broker is responsible for aggregating and maintaining a consistent representation of context.

Author's personal copy

An agent-oriented architecture to support context-aware services in mobile environments is presented in [18,20]. The architecture comprises a multi-agent middleware providing information and context to mobile users. The middleware deals with storing and retrieving information, reasoning based on templates and Case Based Reasoning [1], communication and aggregation of contextual information.

The above examples demonstrate the use of agents as autonomous entities but not as ones which collaborate to enhance their reasoning about context. Existing approaches mostly rely on centralized reasoning services and shared repositories of context descriptions. An advantage of such centralized repositories is common representation and disambiguation of contextual information. Centralized reasoning (as opposed to only keeping a shared vocabulary), however, can become a bottleneck and a single point of failure and can limit the benefits of using the multi-agent paradigm in promoting agility in complex pervasive systems. By maintaining centralized repositories of common vocabulary but moving the reasoning into agents in the decentralized pervasive environment we can gain the following advantages: (1) parallel processing of reasoning, (2) working in the absence of communication links or down-links between agents, or between individual agents and the central service, (3) autonomy and independence: each agent does its own reasoning (with knowledge at hand) when required and as necessary without needing to always lock-step synchronize with others, and (4) robustness, in case of failures (e.g., the centralized service is down). Pervasive systems need to support heterogeneous agents with their own context description, reasoning approaches and goals. Enabling agent collaboration and distributed reasoning about context requires new modelling and algorithms for agents in a pervasive system.

Finally, as an agility mechanism that can be deployed in future pervasive systems we suggest the mobile agent paradigm. Mobile agents are software components that provide agility and re-configurability to a system by being able to move on their own volition or when invited, from one host to another in a fairly autonomous way (i.e., not only is data transferred, but also code and computation state). While not all agents require mobility, this property has been recognized as beneficial in a number of large scale distributed applications, including large networks of embedded systems such as wireless ad hoc sensor networks [35].

Very little work has been done on incorporating mobile agents into pervasive computing systems. We note [44,11,2,3] as first attempts to use mobile agents in such environments.

2.2. *Modelling and reasoning about context*

Effective agent-based middleware supporting reasoning about context under uncertainty requires a cohesive and unifying context modelling; facilitating reasoning that can be applied over the modelled information. We note two major trends in research about context modelling:

(i) Ontology and logic-based models (e.g., [46,8,16,7]), which use logic predicates in describing context. In general, such models provide a way to uniformly represent context, whose structure and semantics are often specified in the ontology of context types. Reasoning is typically applied with some logic-based mechanism over predetermined rules. Logic-based reasoning, however, is typically suitable for dealing with precise

Author's personal copy

information, and inaccuracies in sensed information are not naturally incorporated in most logic-based inferences (exceptions to this include fuzzy logic [47] and probabilistic logic [13]). Rules are often articulated for accurate facts, determined in terms of true or false (not normally associated with uncertainty factors) and are then evaluated in a logical expression. In other words, in determining rules for inferring situations, there is no differentiation regarding the relative importance of facts or a consideration of the accuracy and reliability of these facts.

Although formally depicting contextual information, logic-based models do not and are not originally intended to provide guidelines (or an intuitive approach) for building (or prototyping) context-aware systems. There are no guidelines regarding what relationships to model between entities of the model, what kinds of rules to specify or what predicates to use. In large scale context-aware systems, such an approach can also become difficult to model, and with many application specific rules, it is difficult to gain an overall picture of the context model.

Finally, the use of plain logic predicates and inference rules restrict the ability to treat context as a first-class object of the context model. We distinguish between inferring context using information described with predicates and the ability to model context and changes in context explicitly. With logic-based approaches (unless extended with new abstractions), systems can identify changes in inferred situations but not finer changes in context at any given time, including under the same occurring situation.

(ii) At the other end of the scope, there are a number of applications employing classification and probability-based sensor data fusion techniques as a promising approach for achieving context-awareness (e.g., [12,15,22,23,42]). These approaches look promising for inferring and interpreting context in uncertain environments. However, they either lack an underlying context model, being tailored for particular algorithms rather than represented in a general context model, or in some cases, employ a simplistic model which is suitable for a specific application type or domain. Without an underlying general-purpose context model it would be difficult to develop sufficient flexibility and efficiency, which is required for applying sensor data fusion to changing context related scenarios, dynamically and possibly at runtime. Thus, from a modelling perspective such approaches lack several properties such as being easy to generalize and to use, even though they are amenable to reasoning.

2.3. Modelling and reasoning in the ECORA framework

The ECORA framework adopts the *Context Spaces* model [28,30] to describe the context and apply reasoning over modelled information under uncertainty. The Context Spaces model overcomes some of the shortcomings of logic-based modelling and lack of unifying properties of sensor data fusion approaches for context-awareness.

Context Spaces aims towards a general context model to aid thinking and describing context, and to design operations for manipulating and utilizing context. The concepts use insights from geometrical spaces and the state–space model [27], hypothesizing that geometrical metaphors such as states within spaces are useful to guide reasoning about context. The model provides a unifying way to represent context and enables effective reasoning to be applied over the modelled information. The Context Spaces

Author's personal copy

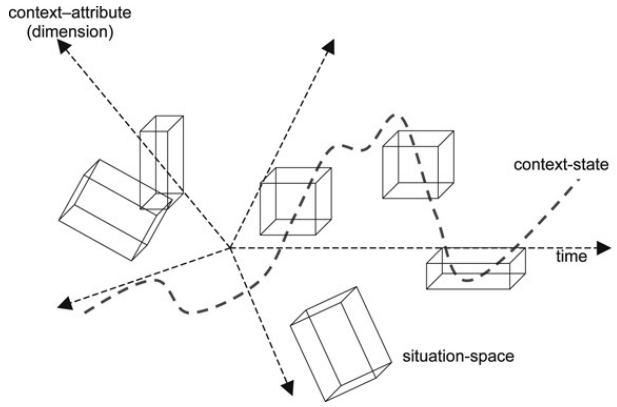


Fig. 1. Illustration of the model's fundamental elements. A collection of situation spaces and the context state, figuratively in multidimensional space, defined over the system context attributes.

approach distinguishes between the concept of context and that of situation. Context is the information used in a model for representing real-world situations. Thus, situations are perceived as a meta-level concept over context, and algorithms are designed to assess the association and mapping between context and situations (i.e., determining occurrences of situations based on information in a model).

In the Context Spaces model, this relationship between context and situations is represented in a general way by the concepts of state and space. The universe of discourse in terms of available contextual information for an application is determined by the types of information, deemed relevant and obtainable by the system designers. We think of a multi-dimensional space made up of a domain of values for each relevant information type, in which context can be sensed. Within such a space we model subspaces (possibly defined in fewer dimensions), which reflect real-life situations. We call these subspaces *situation spaces*. A situation space comprises a collection of *regions*, each region corresponding to a context attribute or the set of values that the attribute can take. (Situation spaces are defined over regions of values in selected dimensions and represent collections of values that reflect real-life situations). The actual values of sensory originated information are defined by the context state, e.g., the collection of current sensor readings representing a specific context.

These fundamental concepts of the Context Spaces model are presented in Fig. 1. Dimensions are defined by context attributes, representing relevant and obtainable types of information. Situation spaces represent specific regions of values in selected dimensions, which reflect situations of interest. For example, in numerical form an accepted region would describe a domain of permitted real values for an attribute, such as the region of values of body temperature between 36.2 and 36.9 °C, representing the range of temperature values of a "healthy person". Or, a collection of non-numerical values such as {'Fine', 'Sunny', 'Partly Cloudy'} which represents a region of acceptable values for the weather conditions context attribute of some situation. The situation space would be defined over a collection of such regions.

The context state fluctuates with respect to time and could occasionally match the definitions of some situation space. For example, a context state at time t could be made

Author's personal copy

up of specific context attributes values such as, say, light-level, noise-level and motion-level. The actual fluctuation of the context state is discrete (representing discrete sensor readings values over time). The context state trajectory, i.e. a curve in space that the system follows over time in a subset of context attributes, can be computed, and related concepts such as the pervasive systems' stability or instability in a given context (i.e. the ability to identify and possibly affect the steadiness of the system state in a given context) become possible.

Based on the spatial representation of context a number of sensor data fusion and reasoning algorithms have been developed [29,32,28,30], including a sensor data fusion approach, which, on the one hand, incorporates heuristics dealing with the relative importance and accuracy of information, and on the other, is applied over information represented in the unifying model. By identifying important properties of situations, the algorithm offers a consistent strategy to model and reason about context and situations under uncertainty.

In reasoning, we follow the idea that states within spaces provide an initial (although, at first, naïve) indication of the occurrences of situations. We extend this notion to model and incorporate during reasoning the significance of particular context attributes and the uncertainty of information (e.g. the uncertain position of the context-state, as measured by inaccurate sensors and how this impacts the confidence in the reasoning outcome). We also incorporate other heuristics using Multi Attribute Utility Theory to compute a utility value indicating our confidence in the occurrence of a situation. Our utility-based approach takes the information represented by the model (i.e. the condition of the context state and the definition of the situation space to be inferred) and computes a confidence (or degree of support) in the occurrence of that situation.

This reasoning approach provides a convenient way for combining together seemingly different contributions into a single measure, expressing the result in terms of utility. Designers of different applications can introduce new heuristics and apply different subsets of such heuristics during reasoning about context. Generally, we see a utility (or contribution towards our goal of determining the occurrence of a situation) as the degree of evidential support given to the hypothesis of a situation occurring when a context attribute value is within the corresponding region. The more indicators we have that the context state matches the definition of a situation space, the greater utility is gained.

We highlight the following advantages of modelling and reasoning using Context Spaces compared with logic-based models and existing sensor data fusion techniques:

(1) The approach offers a consistent strategy to model and reason about context and situations. The concepts of situation space and context state are general and directly used by reasoning.

(2) The model deals explicitly with context (where context is a central concept in the model). Unlike logic-based approaches, it enables representing and analyzing the system state in a given context at any point of time, and computing the confidence with which a situation or a set of situations is occurring.

(3) It naturally incorporates heuristics dealing with uncertainty and can be easily extended to include additional heuristics, or a subset of heuristics, relevant to the application.

Author's personal copy

(4) An overall picture of the context model and the current state of the system in regard to selected context can be gained.
(5) It enables the application of sensor data fusion over a unifying context model.

The modelling approach is further extended with Context Spaces algebra [31,30], comprising operators that enable distributed context reasoning, including compositions of situations and reasoning about logically conditioned situations, in cooperative context-aware multi-agent systems. The algebra facilitates merging between the perspectives of different entities in the pervasive environment, enhancing the reasoning outcome under uncertain conditions, enabling adaptive agent behaviour in multi-agent systems, and computing a numerical measure of confidence in the validity of situation expressions under uncertainty.

3. Architecture of ECORA

The ECORA framework addresses a broad range of issues concerning prototyping context-aware applications, including modelling and reasoning about context by implementing the Context Spaces approach, scalability of reasoning services, effective communication in the pervasive system, and facilitating adaptive and cooperative behaviour of computational entities. The framework provides functionality with a set of Java classes that can be used in building prototype context-aware systems. By starting from a functional component library, we achieve flexibility to facilitate different architectural designs for context-aware computing systems. For example, we start with the notion of a scalable reasoning server, providing services to clients (including mobile agents). However, we use the same functionality to support a decentralized mobile architecture for context-aware computing. In the latter case, agents (e.g. applications, end users, mobile software agents) can individually become context-aware with appropriate functionality and autonomously exchange information.

The framework is conceptually grouped into four aspects, providing services that handle different stages in the life cycle of context-aware computing systems. It enables clients to generically specify context models (i.e., definitions of situation spaces) both at design time and at runtime and to share those models with each other. A number of services, transparent to client applications, make the basic framework. These handle sensory originated data and provide reasoning capabilities for context-aware clients. The framework provides an API and a simple language to describe situations to enable context and situation awareness for clients. The four conceptual aspects (left to right) are presented in Fig. 2.

- The reasoning aspect handles the inference task. It provides the capability to reason about sets of conditioned situations with a built-in implementation of Context Spaces algebra operations and inference algorithms. The reasoning aspect also provides event-based notification based on changes in a given context or predicting such changes.
- The communication aspect provides several communication facilities, addressing various needs in a distributed system. These include message queuing services to handle load, content-based event notification services that decouple senders of notifications from receivers, and point-to-point TCP/IP-based communication between known

Author's personal copy

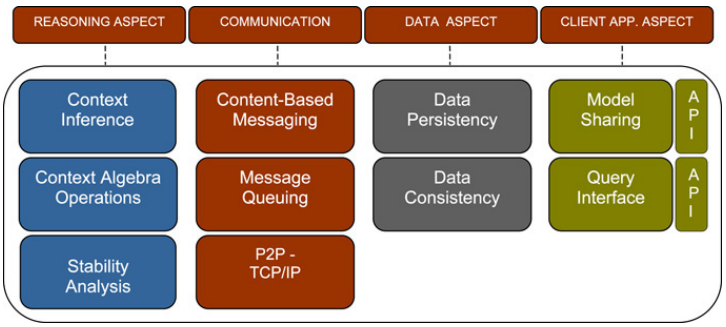


Fig. 2. Four functional aspects in the framework architecture.

entities. The elaborate communication scheme is essential in achieving a scalable, flexible and agile computing platform.

- The data aspect provides services that handle data persistency and consistency. Context-aware environments often produce large volumes of sensed information and the data aspect task is to maintain a repository of consistent and up-to-date information. Historical information that can be used for data mining is maintained, separately. It also provides services to store information, whether it is a sensor reading value or complex context models that are shared by clients.
- The client application aspect provides proxies to the services the framework offers. From a client application viewpoint, there exists only a single interface that represents the entire context framework. A client application, be it a program requiring reasoning services or a program monitoring a collection of sensors, uses the same type of proxy object, which provides it with suitable services. It is the proxy's task to communicate the received requests and information accordingly, and provide services the client requires, in a transparent way.

From a deployment perspective, the four conceptual aspects are translated into services and a set of programmable objects, used by clients. Fig. 3 depicts an overview of the deployed components. We separate the environment into two distinct deployed domains, namely a reasoning context server that supports context-aware computing, and the domain of client applications using the services of the context server. Three separate services are active in the context server. These include (1) the COR Reasoning Kernel, which provides inference services, (2) the COR State Monitor, which provides services for state monitoring and analysis of context, and (3) the COR DB Service, which handles the consistency and persistency of information in the database. The correspondence of Fig. 3 with Fig. 2 is as follows: the context server implements the reasoning aspect, the client application aspect is indicated in Fig. 3; the components communicate via the communication aspect, and the DB implements the data aspect.

Clients make use of COR-Proxy objects, which communicate with appropriate services to fulfil the client's task. Three types of communications are being used, namely, (1) point-to-point — between the proxy object and the reasoning and monitoring services, (2) message queuing — between the proxy and COR DB service (for persistency of sensed information and shared models), and (3) content-based event notifications — between the reasoning and monitoring services and the proxy objects, decoupling senders from

Author's personal copy

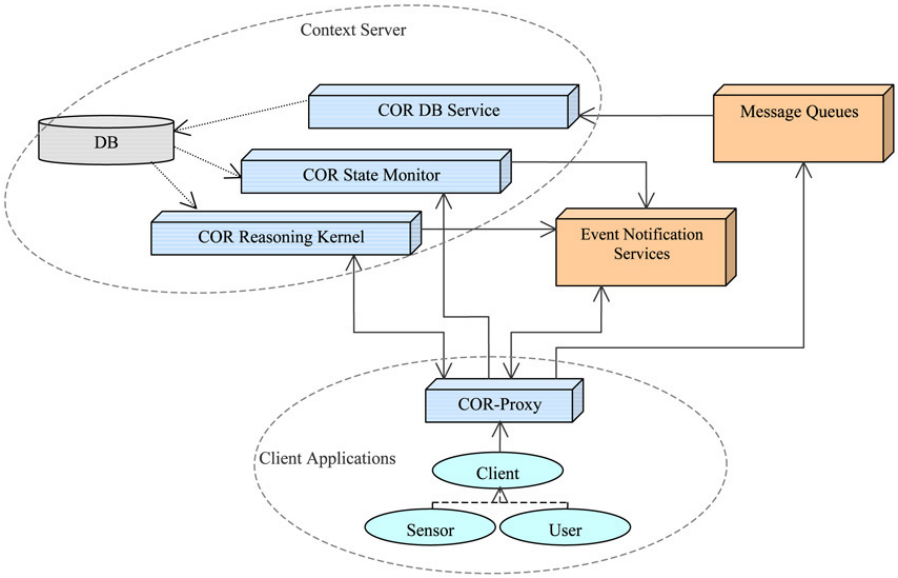


Fig. 3. Overview of deployment architecture.

receivers, such that one entity can request information from others without knowing their precise identity. This type of communication also provides asynchronous messaging for clients that request monitoring services such as notifications about occurrences of specific situations.

The framework provides a functional component library to facilitate flexible construction of context-aware services. Major libraries include the following. (1) The *Model* package, which provides a way to model situations based on the concepts of Context Spaces. The package consists of a collection of classes that represents fundamental concepts, such as context attributes, context state, situation spaces, regions of acceptable values, predicates, and so on. It enables the creation of a hierarchical context model, made up of situation spaces, each defined over a collection of acceptable attribute regions. The package enables clients to create context models, incorporating specific information regarding sensors, context attributes' inaccuracies, and typing of sensed information (e.g., numerical or textual). It stores the received information in data structures, which enable convenient extractions and effective representation of the context description. The set of classes in the Model package is the common means to describe and perceive context related information, used by all framework components and participating entities (e.g., external user applications). (2) The *Kernel* and (3) *Algebra* packages, which provide the core functionality of reasoning about context under uncertainty. Reasoning is performed over context models or situation space objects, defined using the Model package. The classes in this package deal with parsing and decoding complex situation expressions and performing context algebra operations. (4) The *RPrx* package, which provides functionality for creating and deploying proxy objects by the middleware clients.

Author's personal copy

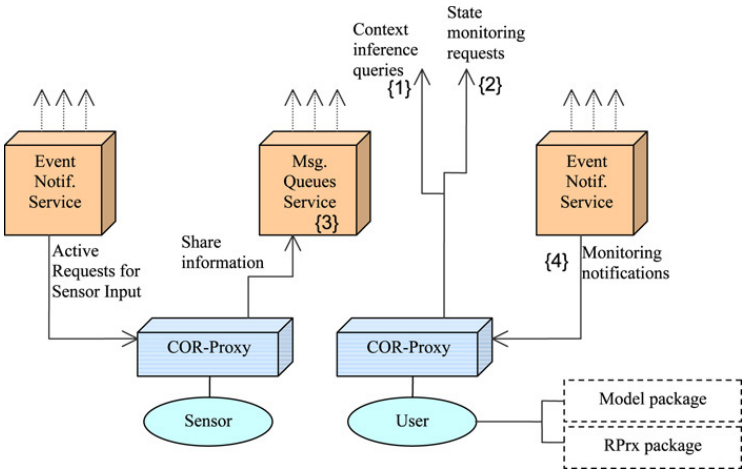


Fig. 4. Routing requests by service type.

3.1. Computing transparency with COR-Proxy and the model package

COR-Proxy objects are used by clients to access the middleware services. The proxy provides an interface to perform various tasks depending on the client's type. For example, clients can request reasoning over complex situation expressions ({1} in Fig. 4) or register to receive events when some situation occurs ({2} in Fig. 4). Alternatively, clients can use the proxy to share information such as sensor readings ({3} in Fig. 4). In general, we distinguish between two broad types of clients, namely, applications that use the proxies for receiving reasoning services, and applications that contribute information into the system, such as applications that monitor sensor readings. Clients that use the proxy can be both of these types at the same time. Requests for different tasks are routed to different appropriate services by the proxy object. This activity is depicted in Fig. 4; the proxy object routes requests according to the specific task, sending it directly to the reasoning components ({1} and {2}), to a message queue ({3}) (which is listened to by the DB COR component), or to the event notification mechanism.

Similarly, replies from services are received by the proxy either synchronously, in which case it forwards the result back to the client method call, or asynchronously, in which case messages are received by the client at a later stage ({4}). Consequently, the proxy approach, which hides the details of the underlying context server framework, provides desirable computing transparency for clients. Together with the ability to create context models, provided by the Model package, clients can participate in a context-aware computing scheme, backed by an underlying context framework.

COR-Proxy provides a Java API for activating system services. It includes a general purpose inference service applied over expressions representing conditioned situation spaces, specified by the client. Major queries include:

- The 'infer' method call provides an answer to queries such as: Is (Situation-A AND NOT Situation-B) OR Situation-C Occurring? The implementation (at the reasoning

Author's personal copy

engine side) supports complex structures, with no practical limitations regarding the expression's size or depth of nested conditions.[1]
- The 'inferByProperty' method call provides the ability to infer the validity of conditioned situations expressions, "pinned down" over some arbitrarily defined property. For example, infer the occurrence of some situation that happens in physical locations, or happens to users (where 'location' and 'user' are property types). The function returns an XML document specifying the validity of all situations that may occur considering the property.
- The 'inferByPropertyValue' infers the validity of conditioned situations expressions, for a specific property value. For example, infer the occurrence of some situation that happens in the 'meeting room' or to a specified user.
- The 'shareInformation' method call shares information such as sensor readings between computing entities.
- The 'monitor' method registers a request for receiving notifications on events occurring in relation to some basic situation. For example, notifications when a user enters and leaves a 'Meeting' situation (where 'Meeting' is defined over a single situation space).

In addition to the API exposed by the proxy object, the Model package also provides an API to build context models and share those models via a shared storage. Fig. 5 demonstrates client code defining predicates for regions of acceptable values as part of the process of defining situation spaces.

In this example, region1 is defined with three predicates over a domain of numerical values between 10 and 50, excluding 11, 22, 33, 44. Similarly, region2 accepts a domain of textual values, assigning a contribution of 1 to elements that are members of the set {'A', 'B', 'C', 'D'}, a contribution of 0.6 to any other element that is not a member of the set {'E', 'F', 'G'} and a contribution of 0 to elements that are members of the set {'E', 'F', 'G'}.

Every instance of a '`Model`' class in the Model package has a method for storing itself (the instantiated class) transparently to a common repository, which is accessed by the reasoning engine. The flow of actions for achieving this is illustrated in Fig. 6. A user application makes use of the RPrx and Model packages to participate in a context-aware scenario. It builds a context model and activates the '`shareModel()`' method to share the locally defined model with other client applications ({1}). The model, which is a hierarchical collection of serializable objects ({2}), is then inserted into storage according to the location of the client application. For example, if the client and database are on different hosts, the model is serialized into remote message queues and then inserted into storage by the COR DB Service ({3}). If both the database and client are local, then the model is directly inserted ({4}).

3.2. The reasoning engine — inference and state analysis

The CORE (Context Oriented Reasoning Engine) represents the two reasoning components residing in the context server, namely, the COR Reasoning Kernel and the COR

[1] Future work extends the language to also include temporal conditions applied over situations.

Author's personal copy


```
double[] bad_values = {11, 22, 33, 44};
Predicate p1 = new Predicate("=<", 50);
Predicate p2 = new Predicate(">=", 10);
Predicate p3 = new Predicate("!", values1);
Predicate[] predicates = {p1, p2, p3};
region1.addSubRegion(predicates, 1);

String[] values1 = {"A", "B", "C", "D"};
Predicate p1 = new Predicate("=",values1);
Predicate[] predicates1 = {p1};
String[] values2 = {"E", "F", "G"};
Predicate p2 = new Predicate("!",values2);
Predicate[] predicates2 = {p2};
region1.addSubRegion(predicates1, 1);
region1.addSubRegion(predicates2, 0.6);
```

Fig. 5. Defining predicates over regions of acceptable values in the definition of situation spaces.

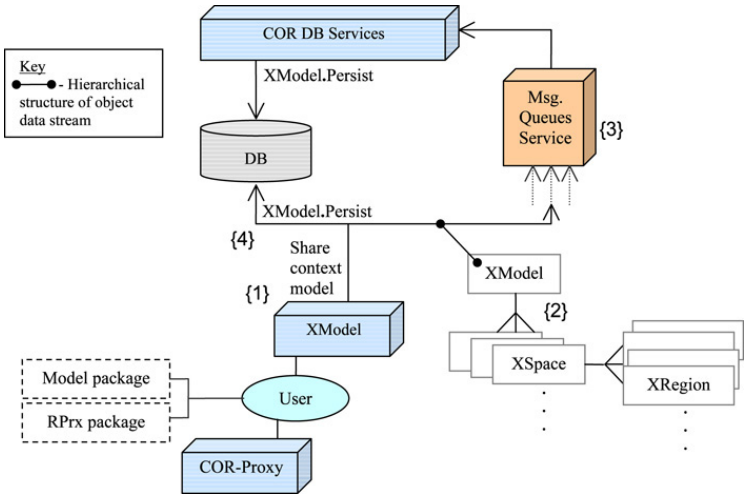


Fig. 6. Sharing context models.

State Monitor. The basic reasoning process is performed for any reasoning request. The complete life-cycle of the reasoning process comprises additional procedures that perform context verifications [30], which partially resolve existing information discrepancies. The latter activity is configurable and optional for any given inference request.

The general activity performed by the reasoning engine triggered by client requests is depicted in Fig. 7. A client proxy object initiates a request, either directly to the COR Reasoning Kernel via TCP/IP requesting reasoning services and synchronously awaits a reply (sequence {1}, {2}, {3}); or it registers to monitoring services from the COR State Monitor and is notified when some context related event occurs (sequence {4}, {5}, {6}, {7}).

The context state information, which is used during the reasoning process, is extracted directly from the local database, residing on the context server. While this information is

Author's personal copy

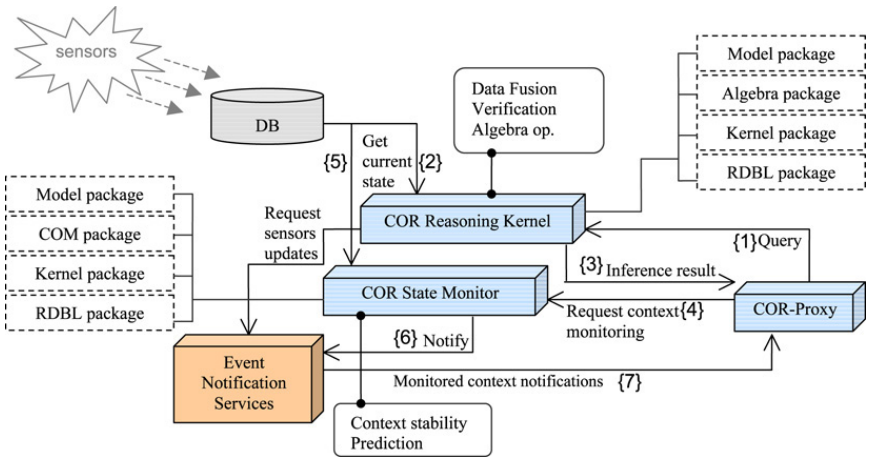


Fig. 7. Reasoning engine (CORE) activity.

considered up-to-date, it is not assumed that it consists of information from all available sensors. The framework allows two types of information retrieval from sensors, namely, active and passive (i.e., push and pull methods), according to the specific sensor design. We assume that application programs that use the proxy object can be used to monitor the sensor readings and/or directly request information from those sensors if such action is feasible.

During reasoning a specific method identifies if any information that is required for the inference is missing. In such cases it actively issues a request for the missing information via the content-based event-driven notification system, specifying the specific kind of information required. The reasoning engine does not deal with the identities of the sensors that may have that information (the sensors are architecturally decoupled from the reasoning engine, having no direct link between them). Instead, applications that monitor sensors are notified of the request and are responsible for sharing such information if it is available to them, via the proxy object.

4. Agent mobility

Client–server architecture used in building context frameworks can arguably deal well with challenges of existing distributed systems. However, as pervasive systems become larger in scale, more open and heterogeneous, and applications more challenging, the autonomy, agility and dependability of distributed components would become critical architectural features. In such conditions, the mobile-agent paradigm becomes most valuable as it supports these features by integrating autonomy with mobility. In complex pervasive systems, the ability to fully exercise centralized control is questionable and hence autonomous lightweight components, managing different tasks for different clients, can become advantageous. Incorporating context-aware mobile agents into pervasive systems would enable agility and fault-tolerance (e.g. in the face of frequent disconnections). Agent mobility would also become useful given an unreliable nature of the underlying wireless network and resource limitations of devices. For example, using mobile agents to track

Author's personal copy

users, utilize the most reliable resources and being close to the source of information in face of recurrent disconnections. These capabilities contribute to the reduction of network load and enhance the system's scalability and fault-tolerance [24,19]. Consider the following scenario as a motivating example for the benefits that can be attained by use of mobile agent technology in context-aware pervasive applications.

Scenario — Emergency evacuation system

Consider an emergency evacuation system in which an assortment of computing devices, such as physical sensors, monitoring applications and software mobile agents in wireless network settings, make up a dynamic pervasive system. During emergency evacuation it is critical to guide employees/residents in a building to safe exits via the shortest and safest routes. With client–server architecture it can become difficult to actively guide each and every individual user (from scalability and physical usability perspectives) who may be trapped in specific locations or encounter specific difficulties en route. For instance, there might only be a limited number of electronic signs along the way to provide essential evacuation information. In contrast, by using the mobile agents' paradigm the system is able to dynamically "embed" intelligence in large numbers of unused devices in the environment. An agent injection service can "inject" mobile agents into devices such as unused desktops, notebooks, or any other device that can host a lightweight agent framework. Mobile agents can exploit the many devices that surround users in future/modern building/office environments. Mobile agents can then actively and autonomously guide evacuating users, by making use of existing devices in the local environment, such as desktop speakers, monitors and even personal PDAs of users. Context-aware mobile agents can reason about safety context, validate and verify context at runtime and distinguish safe routes by applying reasoning and sharing information, communicating with local sensor devices and with other mobile agents in the multi-agent community. Other similar scenarios are possible as in disaster recovery, military battlefields, communities of users with mobile devices in parts of a city, and bushfires.

To achieve such and similar visions our goal is to develop necessary functionality for developing context-aware mobile agents. We perceive the following two approaches to make mobile agents context-aware.

Mobile agents as an application layer

The first approach treats mobile agents as an application layer utilizing the context framework. In such a scheme mobile agents are deployed on top of the context framework using the reasoning engine services to become context-aware. Mobile agents become clients of the framework connecting to the reasoning server via COR-proxy objects. In such a case they enjoy robust reasoning with information utilized by the reasoning engine, originating from disparate sensors, making the most informed inference decision. However, in this approach mobile agents become dependent on other entities, most notably the centralized reasoning engine, in order to gain context-awareness. This has ramifications regarding fault-tolerance, single point of failure and scalability of the mobile-agent-based system. It also weakens the fundamental properties of mobile agents, namely, autonomy and agility, with consequent limitations regarding managing complex pervasive cases.

Mobile agents utilizing context-awareness functionality

Rather than using a centralized reasoning server for attaining context-awareness, mobile agents can themselves carry the library of functionality for performing context reasoning.

Author's personal copy

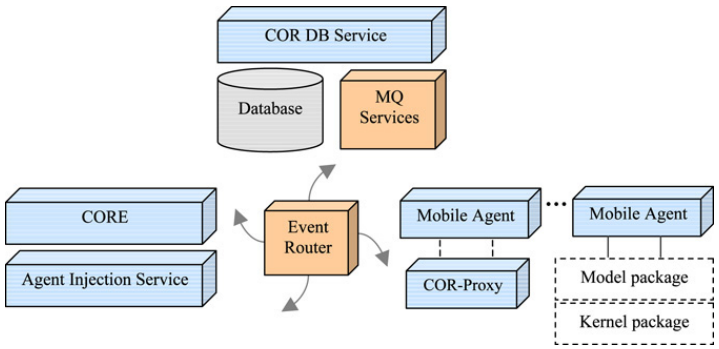


Fig. 8. Proposed architecture in which event notifications are exchanged between context-aware mobile agents, reasoning server and system components.

In such a case, mobile agents use the previously discussed Java packages to build context models and use the reasoning Java package to perform reasoning over the model. This approach, of distributing the reasoning process, embedding mobile agents with their own reasoning capabilities, contributes to a more robust and agile system (see the previously mentioned motivating scenario). Mobile agents with embedded reasoning capability do not need to depend on a centralized reasoning server. However, this approach can also weaken the overall reasoning capability when not all relevant information is locally available. That is, it would be mistaken to assume that an agent by itself has complete information regarding the situations it needs to reason about. In such a case, suitable communication between agents needs to compensate for the lack of information that an agent has, i.e., agents should now effectively share information to gain a broader perspectives of situations.

From the above, we note a trade-off between a centralized "heavy-duty" reasoning approach and lightweight fully distributed reasoning. The combination of both is arguably the most suitable. Agents can be deployed as an application layer on top of the middleware, utilizing robust reasoning services but also capable of reasoning independently.

To make this approach feasible, the ability to share and communicate relevant context information and disseminate events between mobile agents, applications and devices of the pervasive system, needs to be established. With an effective way to communicate, such as with using the publish-subscribe communication genre, facilitating implicit invocations (as noted in [10]) mobile agents could request missing information from other agents without specifying specific recipient identities but only the type of required information.

Fig. 8 presents our proposed architecture integrating the context framework, context-aware mobile agents and event-based communication between entities of the system. Note the injection service, which embeds mobile agents with tasks and itineraries and injects agents into specific physical/virtual locations in the pervasive system.

To realize the above architecture, we develop and evaluate an implicit event-based mobile agent communication mechanism. While standards and practices for mobile agent communication exist, the problem of implicit communication between agents as well as between agents from diverse platforms or environments poses a major challenge [21]. We highlight the complementary nature and the following benefits of

Author's personal copy

using the publish–subscribe communication model for mobile agent operating in pervasive environments:

- The concept of an event is a natural abstraction for the activities within the environment and the interactions among places and agents. The event service only serves as a means of communicating (potentially large volumes of) short messages among agents or to agents about events, in a lightweight manner.
- publish-subscribe middleware provides implicit invocation [10]. This suits a dynamic environment, where A needs not know B's location since B might be continually moving, and A can interact with other components in the environment without knowing their details (e.g., their name, or how many agents there are, thereby providing scalability).
- The publish–subscribe paradigm is independent of agent toolkits — some middleware provides APIs implemented in different languages, e.g., Elvin [40] implements APIs for C, C++, Perl, Python, and Java languages, and so can be used for communication between agents built using different toolkits.
- Performance studies of publish–subscribe systems suggest that they are highly scalable and are efficient means of communication in large distributed systems [4,40].
- We propose an appealing type of messaging that becomes possible by integrating context-awareness, publish–subscribe communication and mobile agents. We coin the term "context-based" messaging to messages that are routed according to the context of the recipient rather than according to its identity [25]. We motivate the ability to send messages to a collection of agents based on their current context, without knowing the precise identities of the agents, and where agent's context is the situation of an agent as is made known by the agent.

We note that reliability of mobile software itself is a large issue which we cannot address within this paper. However, we contend that software that moves can, in fact, improve reliability in the case of anticipated host failures or disconnections. We also note that security is not a concern we address in this paper as the agents move among authorized hosts.

4.1. Designing and implementing event-based communication for mobile agents

We have motivated the use of the publish–subscribe model and in particular content-based messaging for communication between context-aware mobile agents in the framework. We now turn to realization of this idea. For complete account on challenges, motivation and solution for event-based messaging between mobile agents, we refer the reader to [33]. In this paper we use Elvin [40] as a representative of this kind of event-based model and its realization. The principles of the general publish–subscribe model and its use for mobile agents remains, however, the same for other middleware. Elvin provides a content-based messaging between entities, providing the flexibility to operate in a dynamic environment, independently of the need to configure information relating to the recipients of a notification. The notification itself is encapsulated within an object and contains a list of key-value pairs. The notification element supports several data types, such as integers, floating points, strings, etc. A consumer of messages expresses its interest with a

Author's personal copy

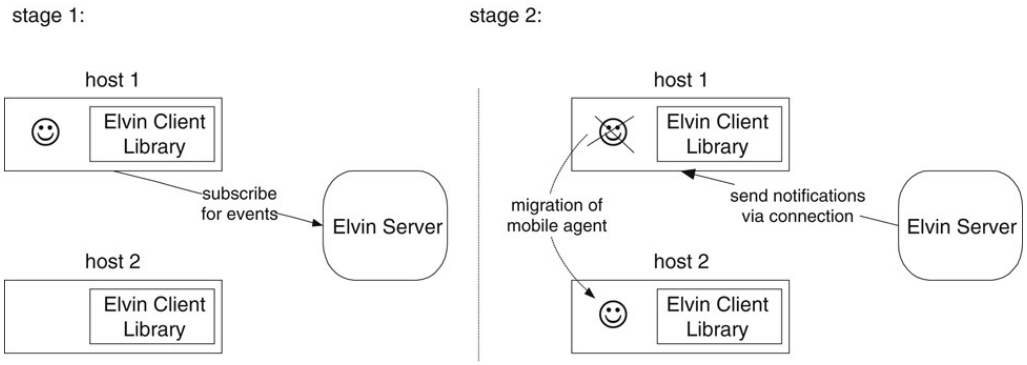


Fig. 9. The problem of publish–subscribe communication with mobile agents.

subscription element, which is built with a special subscription language containing simple logical expressions [49]. Consider the following subscription expression (from [49]):

```
(TICKERTAPE == "elvin" || TICKERTAPE == "Chat") && ! regex(USER,
"[Ss]egall")
```

This subscription expression will match any notification whose TICKERTAPE field has the string value `"elvin"` or `"Chat"` except those whose USER field also matches the regular expression `"[Ss]egall"`. This subscription would match the following notification example:

```
TICKERTAPE:  "Chat"
USER:        "alice"
TICKERTEXT:  "hello sailor"
TIMEOUT:     10
Message-Id:  "07cf0b15003409-5i3N7XDKbPVaQ-28cf-22"
```

Non-mobile programs objects utilizing publish–subscribe mechanisms such as Elvin are static, in the sense that they are located at the same host for their entire lifetime. In contrast, mobile agents are active software entities that migrate between different hosts. During its lifetime, a mobile agent can execute on several agent places at different times and may need to communicate with other mobile agents as well as with static objects. As the focus of all publish–subscribe middleware is to provide an event-based communication for traditional distributed applications and architectures, in which objects remain with the same host, such middleware is unfit to be used as part of mobile code.

There are two problems in integrating publish–subscribe with mobile agents:

(1) The first problem refers to deploying publish–subscribe with programs that change their hosting server (e.g. mobile agents). We illustrate this first problem in Fig. 9, where a mobile agent first subscribes for events (stage 1), and then migrates to another host (stage 2). A communication server contains a subscription registration for the agent, and a connection that is identified with the old location, and so keeps sending new notifications to the wrong host. Hence, new functionality has to be added to facilitate integration of publish–subscribe with mobile agents.

Furthermore, we aim to enable these newly developed capabilities for different mobile agent toolkits and not be restricted to a specific one. The new functionality

Author's personal copy

that supports publish–subscribe communication with mobile code was developed to integrate easily with the regular mobile agent's code, requiring only minimal changes to the agent's program.

(2) The second important mobility problem is concerned with the migration of mobile agents. While in transit, a mobile agent has no concrete existence on a particular host and is therefore unable to receive messages. This problem is intensified in a stateless event notification model, which does not store messages and, therefore, is unable to guarantee the delivery of messages that are lost during transition periods of the agents.

Implementing a solution to problem (1)

To solve the first mentioned mobility problem, new functionality, represented by a `MobileConsumer` class was developed. The `MobileConsumer` class supports Java Serialization, and thus can be transported to different hosts as part of the mobile agent. The functionality implemented in this class follows the approach suggested in [26], in which an agent removes all existing subscriptions before migrating to a new location and re-subscribes to the same notifications after arriving at the new location. The class embeds this kind of behaviour and performs the required operations on behalf of its client (the mobile agent). A `MobileConsumer` object keeps information such as a serializable representation of subscriptions and connection objects. Consequently, a mobile agent client executes the `MobileConsumer` object in two predefined states: before the migration, and right after the migration. Ideally, client code using `MobileConsumer` is implemented in call-back functions that are activated just before and after migration (by the agent's toolkit execution environment). Such a feature of event-based computing is common in most mobile agent toolkits. The following code snippet demonstrates the use of `MobileConsumer` and other Elvin classes in a basic scenario, in which a mobile agent subscribes for notifications:

```
MobileConsumer mc = new MobileConsumer(ServerURL) ;
public void onCreation(Object itin) / init(Object[] creationArgs) {
   Subscription sub = new Subscription(strSubscription) ;
// NotificationHandler → class to handle notification events
   NotificationHandler nh = new NotificationHandler() ;
   sub.addNotificationListener(nh) ;
   mc.addSubscription(sub); }
public void onDispatching(MobilityEvent m) / beforeMove(){
  mc.notifyPreMigration(); }
public void onArrival(MobilityEvent e) / afterMove() {
  NotificationHandler nh = new NotificationHandler() ;
  mc.notifyNewLocation(nh, strSubscription); }
```

('/' separates different names in the Aglet and Grasshopper toolkits for callback methods.)

Implementing a solution to problem (2)

Successful results of the experiment have proven the ability to perform content-based event notification between mobile agents, independently of platform considerations. This has solved the first problem of routing messages to agents that change physical hosting servers. Using the stateless event-based mechanism is lightweight and efficient, even for large volumes of messages, but has one drawback, which is the loss of messages during

Author's personal copy

agent migration (problem (2)). Mobile agents are not responsive during this period (transfer of code and state); hence messages that are sent to them during that time are lost. A relatively long migration time compared with very fast message dissemination implies that a considerable number of messages may be lost during the life cycle of an agent, due to migration. The migration time period is directly affected by communication latency, and more specifically, by the network bandwidth.

Figs. 10 and 11 present experimentation results identifying communication latency and the average stay of an agent in a location as the main factors affecting messages loss. On average, the rate of decrease of lost messages is initially high and gradually declines. The increase in migration delay leads to an increase in the percentage of lost messages. Fig. 12 presents the effects of 8 different simulated migration delays (between 0 and 8) over the average number of lost messages.

Messages loss due to mobility is particularly relevant in pervasive environments in face of degradation in network latency and changes in the agents' behaviour. In one sense, this is not an issue as we are not intending the event-based mechanism primarily as a guaranteed point-to-point communication device but for large-scale event dissemination. Whether a small percentage of lost messages is significant depends on the semantics of the application, and if message loss cannot be tolerated in an application, the problem can be solved by using a stateful, but more heavyweight, event notification system. We are interested, however, in preserving the characteristics of the pure publish–subscribe model, of large volumes of fast small message dissemination and scalable architecture while providing a solution that guarantees message delivery during migration of mobile agents. As a solution that does not compromise these characteristics of the model, we introduce a remote process, residing at the communication server host. The remote process serves as a proxy to the messaging router for mobile agents and temporarily stores messages on behalf of mobile agents during their migration periods.

This approach is similar to [41], an architecture that was designed to provide a solution for disconnecting applications working with Elvin. In our case, we are intending to solve the problem of agent mobility rather than dealing with disconnections, and we leverage on the inherent ability of a mobile agent to be aware of and report changes in its state, prior to migration and after arriving at a new location. The architecture of this solution is presented in Fig. 13. Agents use the `MobileConsumer` object as their interface to perform event-based communication, without concerning themselves with the issue of guaranteeing message delivery during migration periods. `MobileConsumer` then internally communicates with the remote RouterProxy service. When an agent notifies its pre-migration status, the RouterProxy starts local storage of messages that arrive for that particular agent. When the agent notifies that it has arrived at a new location, the RouterProxy checks if messages have been stored, and if so, communicates them back to the `MobileConsumer` object in the same method call.

5. Case studies and evaluation

From theory and architecture we turn to application. We present three case studies, which demonstrate different discussed benefits of using the ECORA framework: (1) we

Author's personal copy

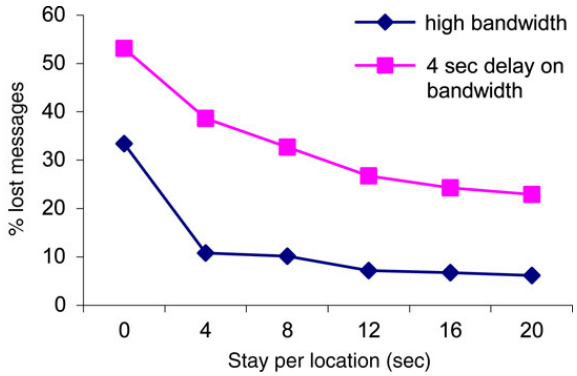


Fig. 10. Effect of changes in bandwidth and length of stay in location over the average percentage of lost messages.

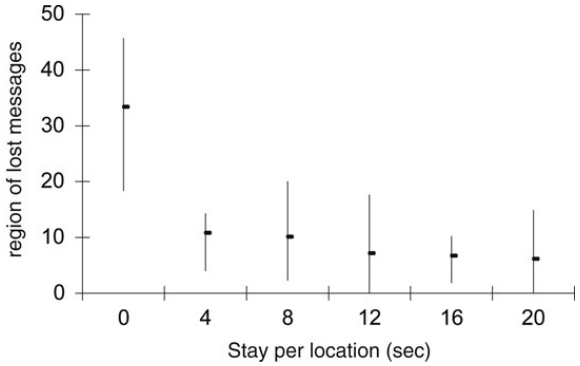


Fig. 11. Aggregated regions of the number lost messages in a high bandwidth experimental run.

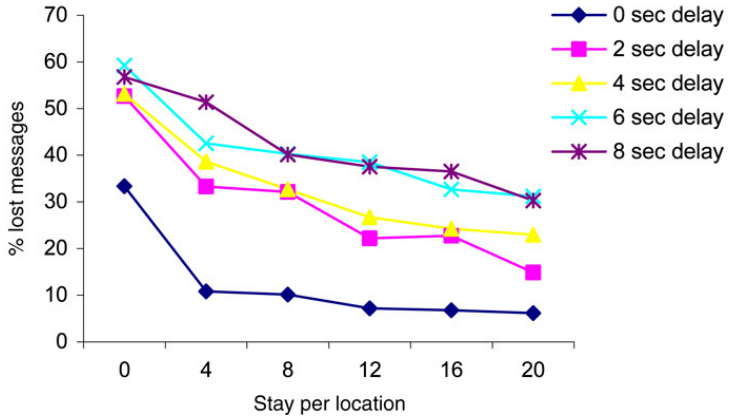


Fig. 12. Effect of changes in duration of stay in hosts and 8 different migration delays.

show the use of the framework and context reasoning engine for prototyping context-based decision making in a unified messaging application (UMA), (2) we demonstrate reasoning via the framework, both about a single situation and a set of conditioned situations, using

Author's personal copy

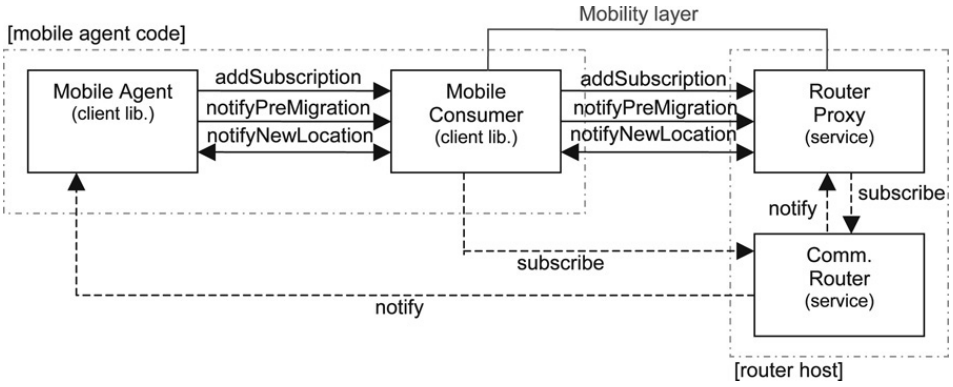


Fig. 13. The architecture for stateless event-based communication for mobile agents including the guarantee of message delivery during migration.

simulation and real sensors, and (3) we demonstrate adaptive and proactive mobile agents, reasoning about the context of a presentation in a smart room.

5.1. A unified messaging application

Unified Messaging Application (UMA) provides a unified messaging platform and relies on context to appropriately route incoming messages to recipients. In UMA scenarios, context can be used to reflect conditions of devices such as mobile phones, notebooks and PDAs, as well as reflect the specific context of users. In a typical scenario, an incoming message to a mobile phone can be converted to text and possibly sent to the intended recipient via email. What determines the actions of the UMA is the context of the devices and users, e.g., the user directs the system to forward messages to his/her email whenever he/she is in a meeting or when his/her mobile phone is switched off or has low battery power. To deal with such preferences the messaging application needs to become context-aware, often reasoning about users' situations under uncertain conditions.

Becoming context-aware with ECORA and prototyping the case of UMA concerns three basic actions. Firstly, relevant situation spaces need to be modelled by application designers. This is a straightforward task concerning situations reflecting devices' status and relevant user situations (possibly described in users' profiles). Secondly, sensory originated information in the system needs to be routed via the framework communication services. Thirdly, inference of conditioned situations' expressions that denote users' routing preferences is performed at runtime by UMA, by interfacing with the reasoning engine via its programming API.

In the following case study, we have produced a simulation for a UMA in an office environment, consisting of 14 processes representing monitor services, each simulating a group of three to five sensors. Each such monitor process produces sensory data for a predetermined logical area, such as an office, meeting room or rest-room. Altogether, we simulate a department with 14 logical areas. Sensors produce randomly generated values within a specified range according to chosen scenarios we test. Values generated are occasionally distorted with significant inaccuracies or may become faulty (i.e., persistently produce wrong readings).

Author's personal copy

The unified messaging application itself services a number of different users, considers their individual messaging preferences and utilizes the middleware for becoming context-aware. UMA utilizes the middleware to forward sensory data, uses the Model package to define relevant situations based on user profiles, and uses reasoning services available by the reasoning engine to determine the context (in order to indicate best message conversion and routing decisions). Consider the following code snippet taken from the UMA client code used in the process of building a model.

```
    // declare a new situation space
XSpace USER_NOTEBOOK_ACTIVE = new XSpace("USER_NOTEBOOK_ACTIVE");
    // define regions for the new situation space
XAttributeRegion a1 = new XAttributeRegion("USER7_NB_KEYBOARD","CONTINUOUS",5,9);
XAttributeRegion a2 = new XAttributeRegion("USER7_NB_ICON","CONTINUOUS",5,9);
XAttributeRegion a3 = new XAttributeRegion("USER7_NB_POWER","CONTINUOUS",5,9);
USER_NOTEBOOK_ACTIVE.addAttributeRegion(a1,0.2,"no");
USER_NOTEBOOK_ACTIVE.addAttributeRegion(a2,0.1,"no");
USER_NOTEBOOK_ACTIVE.addAttributeRegion(a3,0.35,"no");
USER_NOTEBOOK_ACTIVE.addDynamicRegion("USER7_NOTEBOOK_LOCATION",
                            "USER7_LOCATION","=",0.35,"CONTINUOUS");
...
    //declare another situation space and its regions
XSpace MR_MEETING = new XSpace("MR_MEETING");
XAttributeRegion a8 = new XAttributeRegion("MR_LIGHT_LEVEL","CONTINUOUS",7,9);
XAttributeRegion a9 = new XAttributeRegion("MR_NOISE_LEVEL","CONTINUOUS",5,8);
XAttributeRegion a10 = new XAttributeRegion("MR_NUM_PEOPLE","TEXT","3","10");
XAttributeRegion a11 = new XAttributeRegion("MR_MOTION_LEVEL","TEXT","1","3");
...
```

The code demonstrates definitions of situation spaces for a meeting occurring in the meeting room and of a particular user's notebook being active. Definitions require information about context attributes, regions of acceptable values for these attributes, weights reflecting the importance of these attributes to the specific situation, and whether they are symmetrically or asymmetrically contributing to the situation (see [28,29] for more details). Note the `addDynamicRegion("USER7_NOTEBOOK_LOCATION", "USER7_LOCATION","=",0.35,...)` method call, which enables dynamic changes in context attribute values in the region at runtime rather than at design time. In this particular instance, a dynamic region of values is added to the situation of user notebook being active. This prompts the reasoning engine to identify the fact that the locations of the notebook and the user are the same. The reasoning engine performs this check at runtime, testing the validity of the supplied predicate (in this case an equality predicate). If both the notebook and user are inferred to be at the same location then contribution to the occurrence of the situation is added. (Note that the user and notebook locations cannot be predefined.) This action supports the rationale of routing messages to the user's notebook only if it is located close to the user.

Consider the following code snippet taken from the UMA client code used during reasoning about context in regard to user messaging preferences.

```
String expression = "NOT(USER_PC_INACTIVE) | USER_PC_OFF | " + "USER_AWAY_FORM_PC &
USER_MOBILE_ACTIVE " + "& USER_IN_OFFICE_CORRIDOR";
String status = reagine.infer(spaces,expression,null);
...
String status = reagine.infer(spaces,"(MR_MEETING & USER_IN_MR | USER_IN_RR) &
USER_NOTEBOOK_ACTIVE", null);
...
```

Author's personal copy


```
String exp1 = "MR_MEETING & USER_IN_MR | USER_IN_RR";
String exp2 = "USER_AWAY_FORM_PC & USER_MOBILE_ACTIVE & USER_IN_OFFICE_CORRIDOR";
String exp3 =...
expressions[0] = exp1; expressions[1] = exp2;...
Vector result = reagine.compare(spaces,expressions, null);
...
Document doc = reagine.inferByProperty(space1,"GENERAL_MEETING","location");
```

The code snippet demonstrates different reasoning actions performed by the client interfacing with the reasoning engine via the programming API. For example, the client requests reasoning over the complex situation expression:
"`NOT(USER_PC_INACTIVE) | USER_PC_OFF | USER_AWAY_FORM_PC & USER_MOBILE_ACTIVE & USER_IN_OFFICE_CORRIDOR`". Or the expression:
"`(MR_MEETING & USER_IN_MR | USER_IN_RR) & USER_NOTEBOOK_ACTIVE`".

Using the `compare()` method call the UMA client requests the engine to evaluate a series of complex expressions to determine which one is the most supported, computing for each complex expression a numerical confidence value, under uncertainty.

The last example demonstrates using the `inferByProperty()` method call, which instructs the reasoning engine to perform reasoning for a situation that occurs together with some dynamically defined property. In this particular instance, the client requests inference results about meetings that are currently occurring in logical locations and receives an XML document specifying the results for this query. The results of the last query for all meetings currently occurring are presented Fig. 14. Recent requests to the reasoning engine are depicted in the top-right window, an internal representation of the latest reasoned about situation space is depicted in the top-left window, and inference results for the query in XML format are depicted in the bottom window. This format (in XML) is returned to the client, describing in this specific case confidences of having a meeting in each logical area.

The case of unified messaging demonstrates the suitability of the model in describing the context of both human related activities and communication and device conditions. The case study illustrates the relative simplicity of modelling and the resulting effective reasoning, achieved in prototyping context-aware applications using the ECORA framework.

5.2. Reasoning about situations

From prototyping context-aware applications we turn to examine some of the reasoning capabilities used and implemented in the framework for recognizing occurring situations under uncertainty. Firstly, we demonstrate reasoning about the activities taking place in a smart room using real sensors. We apply a sensor data fusion technique [29,32], which considers factors of uncertainty in the environment, such as inaccuracy of sensed information, reliability of sensors and importance of information for inferring particular situations. Our objective is to distinguish between three types of user activities, taking place in a smart meeting room, namely, (1) a user giving a presentation, (2) a user attending another's presentation, and (3) a user attending a meeting. To reason about these activities we have selected four basic context attributes, whose sensors were physically positioned in different locations. These context attributes corresponded to the user location, the meeting

Author's personal copy

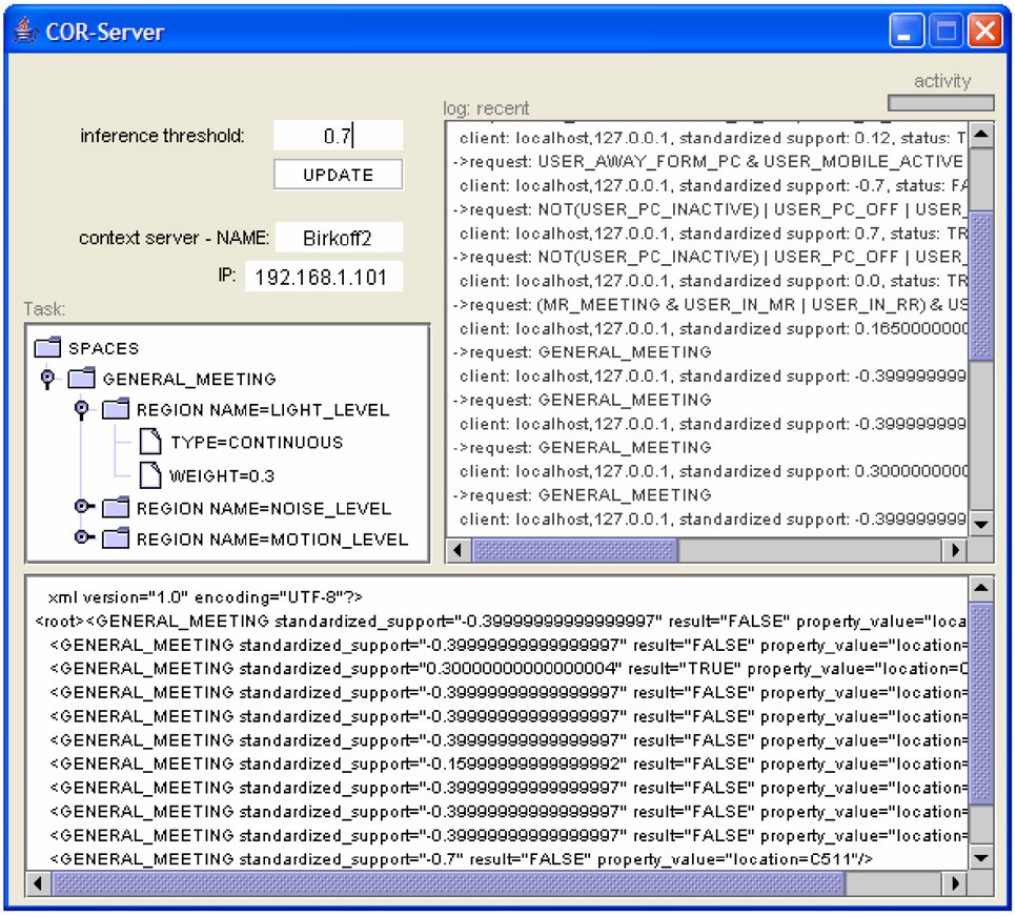


Fig. 14. GUI of the reasoning engine server for the UMA case.

room light level, the user's notebook keyboard and mouse activity and presence of active presentation processes in the user's notebook. For user location we used the Ekahau Positioning Engine [48] that tracks the user's personal devices such as his/her PDA and notebook. The positioning service computes spatial positions by analyzing wireless signal strengths and comparing them to previous calibration. It provides an associated confidence of its inferred location with a measure between 0 and 1. We use Berkeley Motes [48] for sensing and communicating light levels in the meeting room. For retrieving information about the user's presentation activity we have implemented a service that hooks to the notebook operating system and provides information on latest keyboard and mouse activity. We also provide a service that identifies active presentation processes in the user notebook.

During experimentation we have switched between the activities and observed the results of the confidence measures for the different situations. Fig. 15 provides confidence measures obtained for the three activities that were computed for a particular user in two runs.

Results indicate matching confidence levels with the actual activity taking place. Taking the left experimental run as an example, at the time the user is presenting, the confidence

Author's personal copy

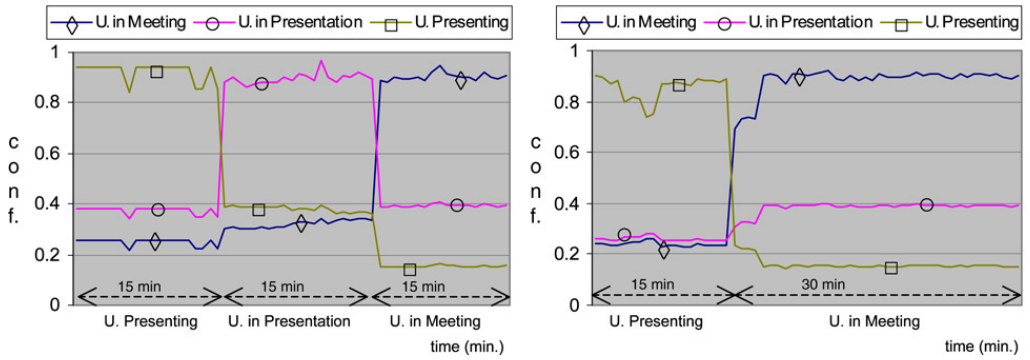


Fig. 15. Experimental runs.

for this particular activity averages around 0.9, whereas confidence in other situations is significantly lower. A change in the situation towards the user only attending another's presentation results in a drop of the confidence in the 'User Presenting' situation to below 0.4 and a rise of confidence in the 'User attending a presentation' situation to levels around 0.9. Similarly, when a discussion (equivalent to a meeting) over the presentations involving the user is starting immediately after the second presentation, the confidence in 'User in a meeting' situation rises to 0.9 and the former situation confidence levels drop significantly.

From reasoning about a single situation we now examine reasoning services of the framework over a set of conditioned situations. The demonstrated reasoning converts complex situation expressions (i.e. logical expressions representing situations) into internal structures held by the reasoning engine and evaluates them while taking into account the uncertainty of the information. The significance of the demonstrated approach is the ability to compute numerical confidence values in the validity of logical expressions (rather than Boolean values) and perform reasoning under uncertainty. This enables evaluation of the occurrence likelihood of mutually exclusive expressions which might both be supported by sensory evidence under uncertainty. In other words, given inaccurate and unreliable sensors, several mutually exclusive situations might be inferred as occurring, and we rely on our evaluation to determine which situation represented by those expressions is most likely taking place.

The reasoning functionality, including logical evaluation of complex situation expressions, was implemented in the kernel package, used by the CORE reasoning engine. In the following experimentation, an application simulating 14 sensors, such as light sensors, location of users, projector status and other physical and virtual sensors, utilizes reasoning services of the reasoning engine, via COR-Proxy.

Randomly generated data for the sensors roughly corresponds to an arbitrarily chosen set of situations that are compatible (i.e. not contradicting by definition). However, simulating inaccurate sensors sometimes yield data which support other situations.

Fig. 16 presents the graphical interface of this demonstration. Different simulated sensors yield different types of information, including numerical, textual and Boolean valued. Sensory originated information is associated with confidence in the reading or, if applicable, knowledge about the range of the reading error (where we assume, in this experiment, uniform distribution of the error). The figure depicts a specific experimental

Author's personal copy

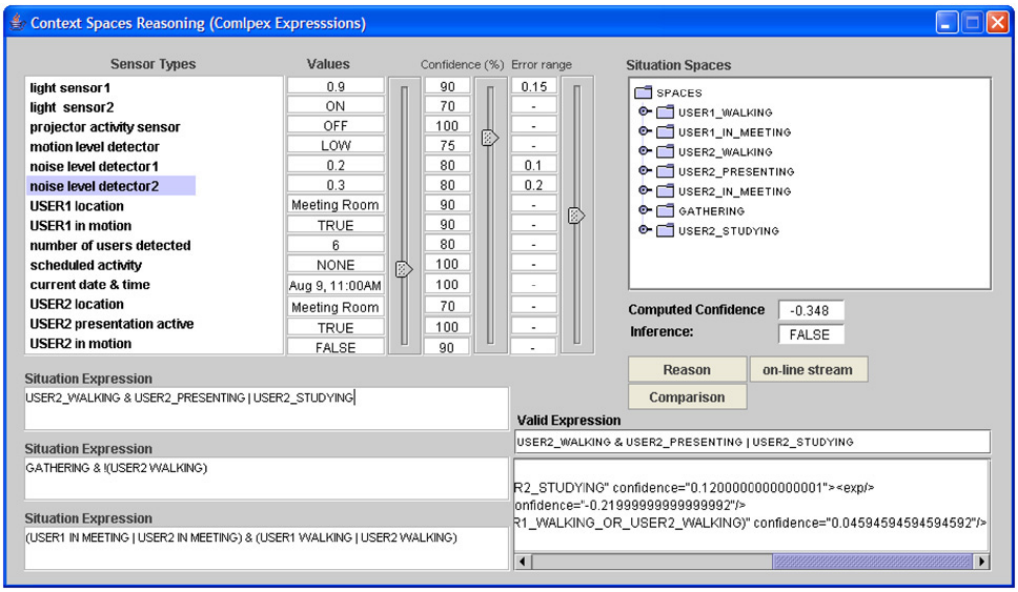


Fig. 16. Evaluating complex situation expressions under uncertainty.

run (also available via applet demo at: http://www.csse.monash.edu.au/~amirp/demo/core-demo6.htm), comparing three expressions:

1. '(User2 walking and User2 presenting) or User2 studying'
2. 'A gathering and User2 not walking'
3. '(User1 in meeting or User2 in meeting) and (User1 walking or User2 walking)'

The occurring expression in this run is expression 1, corresponding to the fact that user2 is studying. Sensory information generally reflects this situation with low noise levels, lights turned on, low motion level of the subject and so on. However, some information also corresponds (although less clearly) to user2 participating in a meeting. Given that user1 is sensed to be moving, expression 2 may also be supported. Since the application defines these expressions as contradictory, it determines the valid expression according to the one with highest computed confidence. This is shown in the bottom right of the figure. Expression 1 is chosen as the most valid; a negative confidence value is computed for expression 2, a positive but low confidence is computed for expression 3, and highest confidence is computed for expression 1 (note that in this evaluation any positive confidence reflects the validity of an expression).

Results of computing confidence for a situation of 'user presenting' given generated sensory information reflecting the user first participating in a meeting, then presenting and finally returning to the meeting, are presented in Fig. 17. Appropriate confidences are computed for reasoning about that situation.

5.3. Adaptive reasoning and agent mobility

Functionality of the framework is utilized in a hybrid architecture, supporting both centralized reasoning via CORE and injecting context-aware mobile agents into the

Author's personal copy

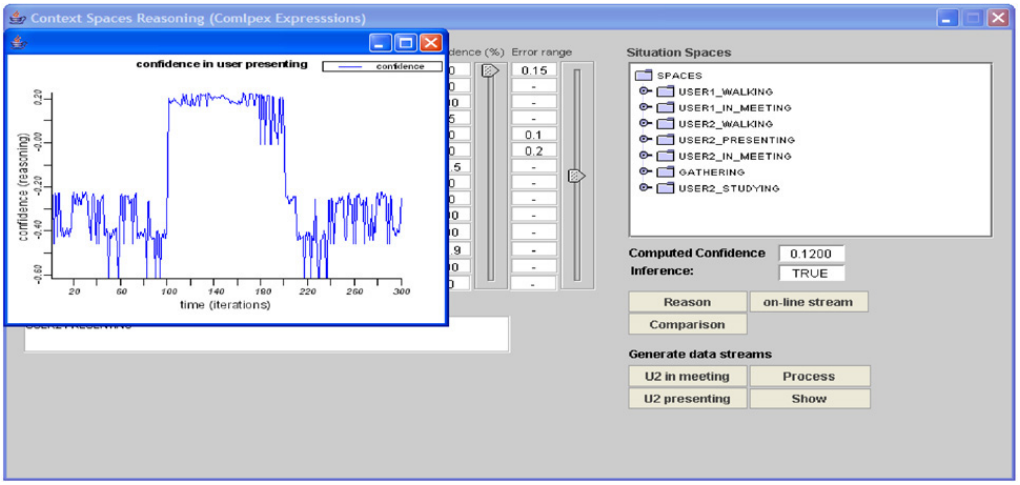


Fig. 17. Demonstrating reasoning about a single situation space.

pervasive system. Mobile agents are embedded with functional components libraries implementing the model, reasoning algorithms and algebraic operations. In addition to enhancing reasoning performance, operations in Context Spaces algebra enable adaptive behaviour of agents in pervasive systems [31]. With our proposed operations, adaptive behaviour of agents, such as active migration between hosts or collaboration between several entities can produce more accurate reasoning. In the following, we have combined Context Spaces modelling, reasoning, and proposed operations with mobile agents as an approach to enable agility and autonomy of context-aware systems.

Our focus here is in applying "merge" and "split" operations. The objective of "merge" is to combine individual context models that reflect the beliefs and capabilities of different agents into a single one, thus producing a stronger representation of the situation, and using more information during reasoning. The objective of "split" or "partitioning" is to produce a new situation model based on a subset of attributes from an earlier situation model (e.g. on a subset of attributes used in the original model that are accessible or known during the time of reasoning). With "split" operation, an agent reasoning about context adjusts its model based on what sensory information is available, so that the updated model only describes information of available sensory data (with recomputed weights reflecting the relative importance between available attributes). This provides the most suitable description of context given a limited subset of sensed/discovered information.

Using the case of presentation activity, we have deployed physically distributed Grasshopper (http://www.cordis.lu/infowin/acts/analysys/products/thematic/agents/ch4/ch4.htm) agent servers. A reasoning mobile agent can migrate between these servers and retrieve relevant sensory information. Each server is logically positioned in a different part of the smart room and is linked to a specific set of sensors. The type and position of sensors affect the way situations are modelled, e.g., different kinds of noise sensors exhibit different sensitivity to noise, and readings obtained from light sensors in the back of the room are different from those obtained in the front. Therefore, different perspectives are produced over the same situation depending on the kind of information presented.

Author's personal copy

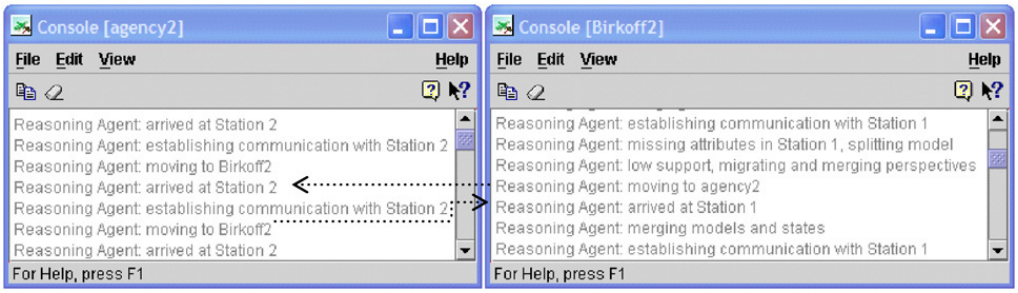


Fig. 18. Agent servers' logs: A reasoning agent proactively migrates to gain additional viewpoints.

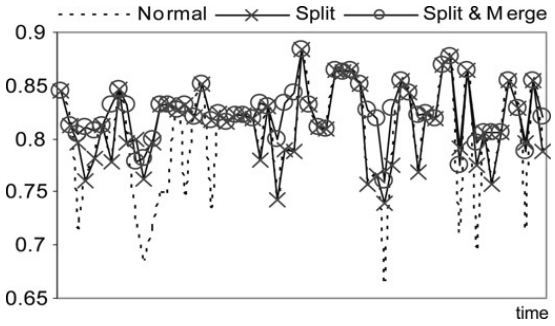


Fig. 19. Effects of partitioning and composing models.

A context-aware mobile agent is positioned in a specific agent server and reasons about the situation using available information. Using algebraic operations the agent is capable of updating its model whenever sensory information is unavailable, and builds a more reliable representation of the situation. In addition, when the confidence about the presentation activity is below a specified threshold, the agent actively migrates to another server in order to take advantage of another perspective (i.e., other contextual information available there). It then merges these perspectives to reach a more reliable inference regarding the occurrence of the presentation. Fig. 18 presents snapshots from the logs of the agent servers, capturing the reasoning agent migration, and performing the split and merge algebraic operations.

Results of this experimentation are presented in Fig. 19. Reasoning with the original model is depicted by the dotted line. Results are improved by updating the model (i.e. restructuring the model into sub-models) according to available information. Finally, the best results (in terms of confidence computed for the presentation situation) are achieved by actively migrating and merging additional perspectives. There are two issues illustrated here: (i) handling partial information by updating the situation description, and (ii) physically migrating to gain additional perspectives over the modelled situation (i.e. through migrating to another location, the agent acquires further information about the same situation – from a different perspective – and so can increase confidence in its conclusions about the occurrence of the situation).

In this experiment, we have demonstrated selected Context Spaces algebra operations and established their usage in an office related use case. Agents dynamically applied

Author's personal copy

compositions and partitioning of situation spaces and performed migration, according to available sensory information and physical access to other perspectives over the situation. We note: (i) a consistent improvement of the reasoning result after applying the proposed operations, and (ii) an incentive for proactive and collaborative behaviour by agents using operations available via the components library of the framework.

6. Discussion and conclusion

We have presented our architectural approach and implementation details for developing a reasoning capable, agile and scalable context framework. We have advocated reasoning and mobility as important ingredients in context-aware pervasive architectures and discussed their use in related scenarios.

We believe that the combination of (1) a unifying context model with algorithms to reason about context under uncertainty, (2) event-based communication as an awareness mechanism, and (3) the ability of components to move as an agility mechanism will be important for the development and evolution of adaptive context-aware pervasive computing systems.

We have developed a functional component library implementing concepts and algorithms of the Context Spaces approach. Based on the library, we have developed the ECORA framework — a flexible, scalable and reasoning oriented framework for context-aware computing. We used the same functionality to provide mobile agents with reasoning capabilities, effectively allowing them to build and reason about context models at runtime.

We have proposed the use of the publish–subscribe communication genre as a complementary form of communication for mobile agents in heterogeneous multi-agent systems. This type of communication is useful in the case of exchanging and disseminating short messages about events and reducing communication complexity between agents. Our approach in concept and architecture is toolkit independent (while we use Elvin as an illustration), and in principle, can be programming language independent, as long as there is support for that language in the messaging middleware.

We have introduced the notion of addressing and messaging agents by the context or situation which they are currently in, which we called context-based addressing. In such a scheme, the publish–subscribe event model can be employed, provided the agents are built to proactively report their context to the event system via subscriptions (in effect expressing their desire to receive messages intended for their present context).

In conclusion, we highlight the following valuable properties of the ECORA framework:

- Providing a unified and expressive description of context and situations by implementation of the Context Spaces model.
- Achieving effective reasoning about context under uncertainty by combining Context Spaces related reasoning algorithms including sensor data fusion and logic-based reasoning, and proposed algebraic operations.
- Achieving (1) flexibility of reasoning about context, (2) agility in approaches to utilize reasoning capabilities, (3) extensibility of the framework and (4) scalability of the proposed reasoning services, by combining client–server and mobile agent design

Author's personal copy

approaches and using a modular functional component library for modelling and reasoning about context.
- Construction of context-aware mobile agents by embedding mobile agents with a lightweight version of the functional component library for individually modelling and reasoning about context and using Context Spaces algebra operations with supported publish–subscribe communication during collaboration.

In future work, we will extend the algebra and support for reasoning about conditioned situation expressions under uncertainty, and further expand the functionality of the CORE reasoning engine and context-aware mobile agents. We will focus on enriching the expressiveness and functionality of the modelling and reasoning components, including providing a lightweight scripting language for reasoning with context spaces, and investigate the impact of mobility over pervasive computing systems.

References

[1] A. Admodt, E. Plaza, Case-based reasoning: Foundational issues, methodological variations, and system approaches, AI Communications 7 (1994) 39–59.
[2] F. Bagci, J. Petold, W. Trumler, T. Ungerer, Ubiquitous mobile agent system in a P2P-network, in: System Support for Ubiquitous Computing Workshop at 6th Annual Conference on Ubiquitous Computing, UbiComp 2004, Nottingham, England, September, 2004.
[3] R.S. Cardoso, F. Kon, Mobile agents: A key for effective pervasive computing, in: ACM OOPSLA 2002 Workshop on Pervasive Computing, Seattle, November 2002.
[4] A. Carzaniga, A.L. Wolf, A benchmark suite for distributed publish/subscribe systems, Technical Report CU-CS-927-02, Department of Computer Science, University of Colorado, April, 2002. Available at: http://www.cs.colorado.edu/~carzanig/papers/cucs-927-02.pdf.
[5] R.H. Campbell, A. Ranganathan, A middleware for context-aware agents, in: Ubiquitous Computing Environments, ACM/IFIP/USENIX International Middleware Conference, Brazil, June 2003.
[6] H. Chen, T. Finin, J. Anupam, An Intelligent Broker for Context-Aware Systems, Ubicomp 2003, October 2003.
[7] H. Chen, T. Finin, J. Anupam, An ontology for context-aware pervasive computing environments, in: Workshop on Ontologies and Distributed Systems, IJCAI-2003, Acapulco, Mexico, August 2003.
[8] A.K. Dey, G.D. Abowd, CyberMinder: A context-aware system for supporting reminders, in: 2nd International Symposium on Handheld and Ubiquitous Computing, in: LNCS, vol. 1927, Springer, 2000, pp. 172–186.
[9] A.K. Dey, D. Salber, G.D. Abowd, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, Human-Computer Interaction 16 (2001) 97–166.
[10] J. Dingel, D. Garlan, S. Jha, D Notkin, Towards a formal treatment of implicit invocation, in: Proc. of the 1997 Formal Methods Europe Conference, 1997. Available at: http://www-2.cs.cmu.edu/afs/cs/project/able/www/paper_abstracts/implicit-invoc-fme97.html.
[11] S. Dobson, Applications considered harmful for ambient systems, in: Workshop on Adaptive Systems for Ubiquitous Computing In International Symposium on Information and Communication Technologies, ISICT 2003, Dublin, 2003.
[12] D. Fox, J. Hightower, L. Liao, D. Schulz, G. Borriello, Bayesian filtering for location estimation, IEEE Pervasive Computing (2003).
[13] J. Halpern, R. Fagin, N. Megiddo, A logic for reasoning about probabilities, Information and Computation 87 (1–2) (1990) 78–128.
[14] A. Held, S. Buchholz, A. Schill, Modeling of context information for pervasive computing applications, in: Proc. 6th World Multi Conference on Systemics, Cybernetics and Informatics, SCI 2002/ISAS 2002, Florida, USA, July 2002.
[15] H.W. Gellersen, A. Schmidt, M. Beigl, Multi-sensor context-awareness in mobile devices and smart artifacts, in: Mobile Networks and Applications, MONET, 2002.

Author's personal copy

[16] P. Gray, D. Salber, Modelling and using sensed context information in the design of interactive applications, in: Proceedings of 8th IFIP International Conference on Engineering for Human-Computer Interaction, EHCI 2001, in: LNCS, vol. 2254, 2001.
[17] M. Khedr, A. Karmouch, ACAI: Agent-based context-aware infrastructure for spontaneous applications, Journal of Network & Computer Applications 28 (1) 19–44.
[18] A. Kofod-Petersen, M. Mikalsen, An architecture supporting implementation of context-aware services, in: Workshop on Context Awareness for Proactive Systems CAPS, Helsinki, Finland, 2005.
[19] D. Kotz, R.S. Gray, D. Rus, Mobile agents: Future directions for mobile agent research, IEEE Distributed Systems Online 3 (8) (2002).
[20] A. Kofod-Petersen, M. Mikalsen, Context: Representation and reasoning, Revue d'Intelligence Artificielle on Applying Context-Management (2005) (special issue).
[21] Y. Labrou, T. Finin, Y. Peng, The interoperability problem: Bringing together mobile agents and agent communication languages, in: J. Ralph Sprague (Ed.), Proc. of the 32nd Hawaii International Conference on System Sciences, IEEE Computer Society, Maui, Hawaii, 1999.
[22] K. van Laerhoven, K. Aidoo, S. Lowette, Real-time analysis of data from many sensors with neural networks, in: Proc. of 5th International Symposium on Wearable Computers, Zurich, Switzerland, 2001.
[23] K. van Laerhoven, Combining the self-organizing map and k-means clustering for on-line classification of sensor data, in: Proc. of Artificial Neural Networks, ICANN '01, Vienna, 2001.
[24] D.B. Lange, M. Oshima, Seven good reasons for mobile agents, Communications of the ACM (March) (1999).
[25] S.W. Loke, A. Padovitz, A. Zaslavsky, Context-based addressing: The concept and an implementation for large-scale mobile agent systems using publish–subscribe event notification, in: Forth IFIP International Conference on Distributed Applications and Interoperable Systems, DAIS 2003, in: LNCS, vol. 2893, Springer-Verlag, Paris, France, ISBN 3540205292, pp. 274–284.
[26] S.W. Loke, A. Rakotonirainy, A. Zaslavsky, Enabling awareness in dynamic mobile agent environments, in: Proc. of the Fifteenth Symposium on Applied Computing, SAC'00, ACM Press, Como, Italy, March 2000.
[27] K. Ogata, State Space Analysis of Control Systems, Prentice-Hall, Englewood Cliffs, NJ, 1967.
[28] A. Padovitz, A. Zaslavsky, S.W. Loke, A unifying model for representing and reasoning about context under uncertainty, in: 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU, Paris, France, July 2006.
[29] A. Padovitz, S.W. Loke, A. Zaslavsky, B. Burg, C. Bartolini, An approach to data fusion for context awareness, in: 5th International Conference on Modelling and Using Context (CONTEXT), Paris, France, in: Lecture Notes in Artificial Intelligence (LNAI), vol. 3554, 2005, pp. 353–367.
[30] A. Padovitz, S.W. Loke, A. Zaslavsky, B. Burg, Verification of uncertain context based on a theory of context spaces, International Journal of Pervasive Computing and Communications (2006) (in press).
[31] A. Padovitz, S.W. Loke, A. Zaslavsky, Multiple agent perspectives in reasoning about situations for context-aware pervasive computing systems, IEEE Transactions on System, Man and Cybernetics (in press).
[32] A. Padovitz, A. Zaslavsky, S.W. Loke, Recognizing and describing situations in sensor-based context-aware applications, ACM Transactions on Information Systems (submitted for publication), (under review).
[33] A. Padovitz, A. Zaslavsky, S.W. Loke, M. Tosic, Agent communication using publish–subscribe genre: Architecture, mobility, scalability and applications, Annals of Mathematics, Computing and Teleinformatics 1 (2) (2004) 35–50.
[34] J. Pascoe, N.S. Ryan, D.R. Morse, Issues in developing context-aware computing, in: Handheld and Ubiquitous Computing, in: LNCS, vol. 1707, September 1999, pp. 208–221.
[35] H. Qi, F. Wang, Optimal itinerary analysis for mobile agents in ad hoc wireless sensor networks, in: 13th International Conference on Wireless Communications, vol. 1, Canada, July, 2001, pp. 147–153.
[36] A. Ranganathan, J. Al-Muhtadi, R.H. Campbell, Reasoning about uncertain contexts in pervasive computing environments, in: IEEE Pervasive Computing, April-June 2004, pp. 62–70.
[37] A. Ranganathan, R.H. Campbell, A middleware for context-aware agents in ubiquitous computing environments, in: ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, June, 2003, pp. 16–20.
[38] A. Ranganathan, R.E. McGrath, R.H. Campbell, D. Mickunas, Use of Ontologies in a Pervasive Computing Environment, in: Knowledge Engineering Review, vol. 18, Cambridge University Press, 2004, pp. 209–220.

Author's personal copy

[39] M. Satyanarayanan, Pervasive computing: Vision and challenges, IEEE PCM (2001) 10–17.
[40] B. Segall, D. Arnold, J. Boot, M. Henderson, T. Phelps, Content based routing with Elvin4, in: Proc. AUUG2K, Canberra, Australia, June 2000. Available at: http://elvin.dstc.edu.au/doc/papers/auug2k/auug2k.pdf.
[41] B. Segall, P. Sutton, R. Arkins, Supporting disconnectedness — Transparent information delivery for mobile and invisible computing, in: IEEE International Symposium on Cluster Computing and the Grid (CCGrid), Brisbane, Australia, May 2001. Available at: http://elvin.dstc.edu.au/doc/papers/invicom01/invicom01.pdf.
[42] H. Wu, M. Siegel, R. Stiefelhagen, J. Yang, Sensor fusion using Dempster–Shafer theory, in: Proc. of IMTC'2002, Anchorage, AK, USA, 2002.
[43] J.F. Sowa, Knowledge Representation: Logical, Philosophical and Computational Foundations, Brooks Cole Publishing Co., Pacific Grove, CA, 1999.
[44] G. Stevenson, P.A. Nixon, R.I. Ferguson, A general purpose programming framework for ubiquitous computing environments, in: Ubisys: System Support for Ubiquitous Computing Workshop, UbiComp, Seattle, WA, 2003.
[45] T. Strang, C. LinnhoffPopien, A context modeling survey, workshop on advanced context modelling, Reasoning and Management, at Ubicomp'04 Nottingham, England, 2004.
[46] X.H. Wang, D.Q. Zhang, T. Gu, H.K. Pung, Ontology based context modeling and reasoning using owl, in: Workshop on context modeling and reasoning at the 2nd IEEE Conference on Pervasive Computing and Communications (PerCom2004), Orlando, FL, USA.
[47] L. Zadeh, Outline of a new approach to the analysis of complex systems, IEEE Transactions on System, Man and Cybernetics 3 (1973).
[48] Information on Ekahau Positioning Engine. Available at: http://www.ekahau.com. Information on Berkeley Motes available at: www.tinyos.net/, www.xbow.com/.
[49] The Elvin Subscription Language, DSTC: http://elvin.dstc.edu.au/doc/esl4.html.

Amir Padovitz is an applied researcher and a developer at Microsoft, Redmond, USA. He was a Ph.D. candidate in the school of Information Technology at Monash University, Australia, and a recipient of Hewlett–Packard Ph.D. endowment for his research. His Ph.D. thesis focused on context modelling and reasoning about situations in pervasive computing systems. He is also a member of the Centre for Distributed Systems and Software Engineering at Monash University. Amir has authored about 30 research publications and industrial patents in the areas of context-aware pervasive computing and web search technology. His research interests include context-aware pervasive computing, machine learning, and web and information mining. He can be contacted via email at amirp@microsoft.com.

Dr. Seng W. Loke is a Senior Lecturer at La Trobe University, Melbourne, Australia. He is also an Honorary Associate of the Center for Distributed Systems and Software Engineering at Monash University. He co-leads the Pervasive Computing Interest Group at La Trobe, and has authored more than 130 research papers. He completed his Ph.D. at the University of Melbourne. He can be contacted at the Department of Computer Science and Computer Engineering, in La Trobe University, or email him at s.loke@latrobe.edu.au.

Arkady Zaslavsky is an Associate Professor at Monash University, Australia. He is also a Professor (Adjunct) at the Lulea University of Technology, Sweden. He received M.Sc. in Applied Mathematics majoring in Computer Science from Tbilisi State University (Georgia, USSR) in 1976 and Ph.D. in Computer Science from the Moscow Institute for Control Sciences (IPU-IAT), USSR Academy of Sciences in 1987.

Arkady Zaslavsky has published more than 200 publications throughout his professional career. He has organized and chaired many workshops and conferences in mobile computing area, including "Mobility in Databases and Distributed Systems" and International Conference on Mobile Data Management, MDM2003. He is an editorial board member for Elsevier journals "Computer Communications" and "Information

Author's personal copy

and Software Technology". His research interests include mobile and pervasive computing; distributed and mobile agents and objects; wireless networks; distributed computing and database systems; and distributed object technology and mobile commerce.

Arkady Zaslavsky has been awarded and involved in many research grants and projects including DSTC's "M3: Enterprise Architecture for Mobile Computation", "Context-rich mobile agent technology to support information needs of financial institutions", "Adaptive Distributed Information Services", "Mobile City" and others. He is a member of ACM and IEEE Computer and Communication Societies.