

Agent Communication in Ubiquitous Computing: the Ubismart Approach

Jurriaan van Diggelen, Robbert-Jan Beun, Rogier M. van Eijk, Peter J. Werkhoven
Institute of Information and Computing Sciences
Utrecht University, the Netherlands
{jurriaan,rj,rogier}@cs.uu.nl; peter.werkhoven@tno.nl

ABSTRACT

This paper studies the use of agent communication in ubiquitous computing. This application domain allows us to investigate the efficient handling of large quantities of information in agent-based systems. We will present an approach to dynamically set up a communication network between agents which aims to minimize the communication load. The approach is based on a formal ontological notion of informativeness, on quantitative measures such as information gain and on the proper use of interaction mechanisms such as Publish/Subscribe. We also present experimental results which have been obtained using our prototyping tool called Ubismart.

Categories and Subject Descriptors

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—*multiagent systems, coherence and coordination*;
C.2.4 [Computer-Communication Networks]: Distributed Systems—*distributed applications*

General Terms

Design, Languages

Keywords

Ubiquitous computing, Agent communication, Ontologies

1. INTRODUCTION

Over the last decade, multi-agent systems (MAS's) have had a profound impact on many related research fields among which ubiquitous computing [17]. In the early nineties, Mark Weiser envisioned ubiquitous computing as hundreds of networked computing devices and sensors working together and assisting people in their everyday lives [21]. Since then, much research has been conducted on this new computing paradigm and many grand challenges have been identified, such as preserving privacy, dealing with trust and obtaining interoperability. In this paper we will be concerned with the interoperability issues of ubiquitous computing, which we will study from an agent perspective. The key challenge is to achieve *serendipitous interoperability* [10], i.e. the ability of software systems to discover and utilize services they

Cite as: Agent Communication in Ubiquitous Computing: the Ubismart Approach, J. van Diggelen, R.J. Beun, R.M. van Eijk, P.J. Werkhoven, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX. Copyright © 2008, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

have not seen before, and that were not considered when the systems were designed.

The difficulty of serendipitous interoperability is that it excludes the possibility to fix in advance the communication network between the system components (which we call agents from now on). It is impossible to specify at design-time which agent communicates which information to which other agent. Therefore, we only require the information needs of an agent to be specified at design-time and let the agent find out at run-time how to obtain this information. The agents must have the right communicative skills to set up a communication network between themselves which enables them to exchange sufficient information. However, the agents should not exchange more information than necessary. This is because of three reasons. Firstly, when vast amounts of data are available, information overload becomes a serious issue due to limited storage and computing resources. Secondly, because the system often consists of mobile devices with limited energy supply, mobile communications, which are heavy on power consumption, should be minimized [8]. Thirdly, the system may also contain humans which are even more easily prone to information overload than computers. Thus, the problem we will tackle in this paper can be summarized as follows: how can agents dynamically set up a communication structure between themselves which allows them to exchange sufficient information with as few messages as possible?

Our solution builds partly on related work from qualitative AI. We will show that, by using a layered structure of *ontologies* that are formalized in logic, the agents can apply ontological reasoning to derive what information they can obtain from whom. Another part of our solution builds on related work from quantitative AI, in particular *decision trees* and *probability theory*. We will present three efficiency measures which the agent can use to reduce the communication load in the system.

As a first measure, we will apply decision trees to determine the order in which the different pieces of information should be acquired. Secondly, we will discuss an efficient use of the Query and the Publish/Subscribe mechanisms. Both of these interaction mechanisms are well known within the MAS-community due to agent communication languages such as KQML [4] and FIPA ACL [5]. Our application of MAS's to ubiquitous computing allows us to formulate computational criteria for choosing between querying and subscribing, which, to the best of our knowledge, is a novelty in the literature. As a third efficiency measure, we will introduce a special kind of subscription, i.e. the *conditional*

subscription by which an agent requests to be notified about something when some condition is true.

To validate the ideas introduced in this paper, we have developed a prototyping tool, called Ubismart. This allows us to easily perform agent-based ubiquitous computing simulations and experiment with different communication strategies and OWL ontologies [15]. We will present a simulation experiment in the domain of crisis management. In this system, information from different sensors must be combined to inform a crisis worker via his or her PDA about the potential risks of fire and traffic accidents. By comparing the different communication loads that result from applying different efficiency measures, we demonstrate that the techniques discussed in this paper are successful.

The paper is organized as follows. In Section 2 we will introduce layered ontologies. In Section 3, we will discuss the measures we propose in the communication mechanism to reduce the information flow between the agents. Our implementation and the results of the experiments are presented in Section 4. Section 5 concludes the paper and gives directions for future research.

2. LAYERED ONTOLOGIES

A common ontology has been advocated as the key to achieve mutual understanding between agents as it guarantees that every agent uses the same terms to represent the same meanings [7]. In ubiquitous computing, however, different components typically represent their information at different levels of abstraction, i.e. they view the world differently. These different world-views lead to heterogeneous ontologies which could cause misunderstandings [19]. To deal with this problem, we propose a system with *layered ontologies* [20] where the agents only have parts of their ontologies in common. In this way, every agent maintains its own ontology tailored to its task, and use the common parts of their ontologies for communication.

As a running example throughout the paper we introduce a simple system where a PDA notifies its user about the presence of a *photo moment*, i.e. a scenic sky. The PDA (Ag-6) deals with the high-level concept *photo moment*. Other components (Ag-4 and Ag-5) have less abstract information such as *rainbow* and *sunset*, whereas the sensors (Ag-1, Ag-2 and Ag-3) possess low-level information about light-conditions or the presence of rain or sunshine. The ontologies of the agents are shown in Figure 1.

An agent’s ontology is composed of several contexts, which are related by mappings that specify translations between them, analogous to a multi-context system [6]. Each concept (e.g. c, d, e) is prefixed with a context-name (e.g. $c1, c2, c3$) to indicate to which context the concept belongs. For example, the ontology of Ag-2 contains two concepts c (*rainy*) and d (*sunny*) which both belong to context $c1$. As indicated by the incoming arrow, Ag-2 is a sensor that measures information about concept $c1:d$ (*sunny*). The ontology of Ag-4 contains concepts from three contexts $c1, c2$ and $c3$. Ag-4 relates contexts $c1$ and $c2$ with $c3$ by a mapping, which is indicated by a grey layer.

We assume that a mobile ad-hoc network exists which allows agents to discover the presence of each other, to view which contexts are contained in each other’s ontologies, and to send data to each other. Communication in the system is initiated by an agent aiming to resolve its information needs. Suppose, for example, that Ag-6 has information need $c5:j$

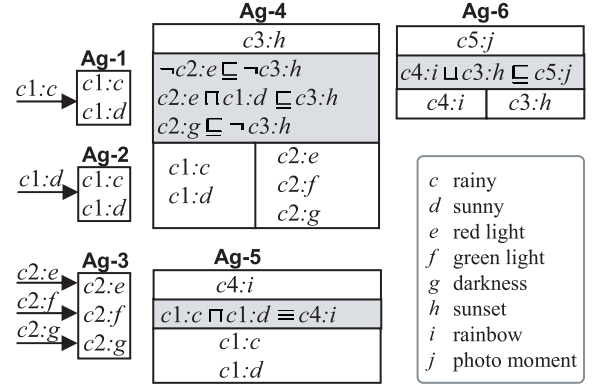


Figure 1: Example System

(*photo moment*). It starts by looking for other agents that can provide valuable information for this concept. Because no other agent in the system has $c5$ defined in its ontology, Ag-6 translates concept $c5:j$ to a lower level concept $c4:i$ (*rainbow*) and $c3:h$ (*sunset*) using its mapping. $c3:h$ can be queried to Ag-4, as Ag-4 is familiar with context $c3$. This query raises the information need $c3:h$ for Ag-4, which it tries to resolve using the same strategy, i.e. translating it to the lower-level contexts $c1$ and $c2$ in order to pass the query on to Ag-1, Ag-2 or Ag-3. Because these agents are sensors, they can obtain their information directly from the world, and the chain of queries ends there.

Crucial in this approach is that an agent must be capable of translating a representation in a non-common context to a representation in a common context to be understood by the recipient. Before we explain this process in detail, we briefly introduce the knowledge representation formalism.

2.1 Knowledge Representation Formalism

An agent’s knowledge base is formalized in description logic and consists of a TBox and an ABox [2]. The TBox of an agent implements its ontology and stores concepts and their relations. The ABox stores sentences constructed using these concepts. The ABox is assumed to be initially empty and grows as the agent senses its environment or communicates with other agents. In Figure 1, only the agents’ TBoxes are shown.

Without going into the formal semantics of description logic, we briefly discuss its constructs. Concepts are composed using atomic concepts and concept constructors, i.e. \sqcap (conjunction), \sqcup (disjunction), \neg (negation). For example, $c \sqcap \neg d$ refers to the concept *rainy* and not *sunny*.

Concepts are interpreted as subsets of a domain of discourse, denoted by Δ . Because time plays an important role in our application, we assume that Δ is a set of time points $\{t1, \dots, tn\}$. For example, if the concept *rainy* is interpreted as $\{t3, t5\}$, it means that it was rainy at time points $t3$ and $t5$. We use a special variable NOW to denote the current time point. This can be implemented by adopting one central time reference for all agents which instantiates the agents’ NOW variables with the current time.

The TBox is specified as a number of inclusion axioms of the form $c \sqsubseteq d$, meaning that the interpretation of c is a subset of the interpretation of d , i.e. all instances of c are

also instances of d . For example, the TBox axiom in the mapping of Ag-4, $c_2:e \sqcap c_1:d \sqsubseteq c_3:h$ means that if there is *red light* and it is *sunny* at some time point, then there is a *sunset* at that time point. The ABox is specified as a number of membership assertions of the form $c(t)$ meaning that t is an instance of c . For example, the ABox assertion $Rainy(t_4)$ means that it is rainy at time point t_4 . We will write $KB \models$ to state that "from the TBox and ABox follows that". We say that an agent has satisfied its information need on a concept c , if $KB \models c(\text{NOW})$ or $KB \models \neg c(\text{NOW})$. For $c(t)$ we sometimes simply write that c is true at time t and analogously, for $\neg c(t)$ that c is false at time t . For $c(\text{NOW})$, we sometimes simply write that c is true.

3. COMMUNICATION MECHANISM

The central issue addressed in this section is how to resolve an agent's information need as efficiently as possible. Firstly, we will define which concepts from other contexts qualify as informative w.r.t. the agent's information need. Then we will discuss what is the best order to obtain information about these concepts. Finally, we will argue by means of which interaction mechanism this information can best be obtained, i.e. by query, subscribe or conditional-subscribe.

3.1 Informative Concepts

The first feature we will introduce in our communication mechanism is intended to prevent queries from being posed which are not informative for resolving the information needs. This is also a very basic aspect of human communication. For example, an investigator attempting to determine the cause of a house fire, will likely inquire about the presence of inflammable material or the smoking habits of the occupants, rather than inquire about the color of the walls. For communicating software agents, precise rules are needed to implement this seemingly trivial property. In description logic, we can specify when a concept in one context qualifies as informative for a concept in another context.

Clearly, a concept $c_i:c$ qualifies as informative for concept $c_k:e$ if membership of concept $c_i:c$ either implies or precludes membership of concept $c_k:e$. This is when $c \sqsubseteq e$, or when $c \sqsubseteq \neg e$. Sometimes, membership of a concept can only be decided by posing multiple queries to different agents, a process known as *query dissemination* [11]. For example, consider Ag-4 in Figure 1. Because $c_2:e \sqcap c_1:d \sqsubseteq c_3:h$, the truthvalue of h can be decided by querying e to Ag-3 and d to Ag-2. Therefore, a concept can also be qualified as informative if it can be regarded as part of what must be known to exclude or conclude membership of the target concept. Returning to our example with the fire investigator, it is justified to ask about the smoking habits of the occupants, because the answer, *together with some other information*, can explain the cause of fire. Formally, this is defined as:

DEFINITION 1. *Informative*

Concept $c_i:c$ is informative for concept $c_k:e$ iff there exists $c_j:d$ for which

- $(KB \models c \sqcap d \sqsubseteq e \text{ or } KB \models c \sqcap d \sqsubseteq \neg e)$, and
- $(KB \models d \not\sqsubseteq e \text{ and } KB \models d \not\sqsubseteq \neg e)$

This definition states that a concept c is informative for concept e , if two conditions hold. The first condition states that, together with some other concept d which stems from

any context, c and d must imply e or $\neg e$. The second condition states that membership of d alone should not imply e or $\neg e$. Hence, the information about c is really necessary for the conclusion. Note that, when concept c by itself is sufficient to imply e or $\neg e$, then c also qualifies as informative. This can be easily shown by taking for concept d , concept \top (which is defined as the superconcept of all concepts).

An agent that queries a concept c does not know whether the answer will provide information that c or that $\neg c$. Therefore, if only one of the concepts c or $\neg c$ is informative for the agent's information need, a query on concept c is allowed.

EXAMPLE 1. Consider Ag-4 with information need $c_3:h$ in Figure 1

- Concepts $\neg c_2:e$, $c_2:e$, $c_1:d$ and $c_2:g$ are informative for concept $c_3:h$.
- Ag-4 may query $c_2:e$, $c_1:d$ and $c_2:g$. Ag-4 may not query $c_1:c$ or $c_2:f$.

The idea of querying informative concepts is similar to backward chaining in expert systems [16]. To know the truth-value of a consequent, all truth-values of the conjuncts in the antecedent must be known. We would call all these conjuncts informative. Having described which concepts are suitable candidates for querying, we will now describe the order in which these concepts should be queried.

3.2 Decision Trees

Not every answer resolves an agent's information need. For example, if Ag-4 queries the informative concept g to Ag-3 and gets $\neg g$ as a response, it can neither derive h , nor $\neg h$ leaving its information need unsatisfied. Therefore, it is useful to anticipate on the expected answer when deciding which of the informative concepts to query first. This is also a common pattern in human communication. For example, a fire investigator usually starts by examining the most frequently occurring causes, such as smouldering cigarette butts, and postpones investigating the rarer causes such as lightning strike. To apply these ideas to our communication mechanism, we use a quantitative measure called *information gain* [13] that indicates how much closer the agent gets to satisfying its information need by querying a certain concept.

Information gain is defined in terms of a measure from information theory, i.e. *information entropy*. This measure can be used to indicate how certain the agent is about the truth-value of some concept. For example, consider Ag-4 with information need h . If Ag-4 has completely satisfied its information need, it is certain that $h(\text{NOW})$ or it is certain that $\neg h(\text{NOW})$, in which case the entropy is 0. If Ag-4 does not have a clue about the truth value of h , the entropy is 1. In this case, the agent's experience does not provide any indication of the truth value of h , i.e. h has been true at half of the time points and false at the other half. This is formalized as:

DEFINITION 2. *Information Entropy*

Given an information need c , and a set time points Δ $Ent(\Delta) = -p \log_2 p - n \log_2 n$, where

- p is the proportion of time points in Δ which are member of c .
- n is the proportion of time points in Δ which are not member of c .

Information gain is defined as the expected reduction in entropy that results from obtaining the truth-value of a concept. An agent that aims to satisfy its information needs tries to get the entropy on 0 as fast as possible. It therefore chooses the concept with the highest information gain. In the definition below, # is used to denote the number of instances in a set.

DEFINITION 3. Information Gain

Given an information need c , and a set time points Δ
 $Gain(\Delta, d) = Ent(\Delta) - \frac{\#\Delta^d}{\#\Delta} Ent(\Delta^d) - \frac{\#\Delta^{-d}}{\#\Delta} Ent(\Delta^{-d})$, where

- Δ^d is the set of domain individuals which are member of concept d .
- Δ^{-d} is the set of domain individuals which are not member of concept d .

By repeatedly ordering the concepts according to their information gain, a tree can be constructed which tells the agent the order in which concepts must be queried. This boils down to the ID3 algorithm for decision tree learning [13], but used in an entirely different domain. Contrary to ID3, we only use information gain for efficiency. The final outcome is based on the logical rules, and not on the set of training examples.

EXAMPLE 2. Consider Ag-4 in Figure 1 with information need h . Consider the following ABox information (note that Ag-4 does not know the columns of concepts i and j):

	c	d	e	f	g	h	i	j
$t1$	F	T	F	T	T	F	F	F
$t2$	T	F	F	T	T	F	F	F
$t3$	F	T	T	F	F	T	F	T
$t4$	T	F	F	F	T	F	F	F
$t5$	T	F	F	T	T	F	F	F
$t6$	T	F	T	F	F	F	F	F
$t7$	F	T	F	F	T	F	F	F
$t8$	T	F	T	F	F	F	F	F
$t9$	T	F	T	T	T	F	F	F
$t10$	T	F	T	F	F	T	F	T
$t11$	T	F	T	F	F	F	F	F
$t12$	T	F	T	T	T	F	F	F
$t13$	T	T	F	T	F	F	T	T
$t14$	T	F	T	T	T	F	F	F
$t15$	F	T	F	F	T	F	F	F

Using this information, we can calculate the following:

- $Gain(\Delta, e) = Ent(\Delta) - \frac{8}{15} Ent(\Delta^e) - \frac{7}{15} Ent(\Delta^{-e}) = (-\frac{2}{15} \log_2 \frac{2}{15} - \frac{13}{15} \log_2 \frac{13}{15}) - \frac{8}{15} (-\frac{2}{8} \log_2 \frac{2}{8} - \frac{6}{8} \log_2 \frac{6}{8}) - \frac{7}{15} (-\frac{0}{7} \log_2 \frac{0}{7} - \frac{7}{7} \log_2 \frac{7}{7}) = 0.567 - 0.433 - 0 = 0.134$
- $Gain(\Delta, d) = Ent(\Delta) - \frac{5}{15} Ent(\Delta^d) - \frac{10}{15} Ent(\Delta^{-d}) = 0.567 - 0.24 - 0.312 = 0.015$
- $Gain(\Delta, g) = Ent(\Delta) - \frac{9}{15} Ent(\Delta^g) - \frac{6}{15} Ent(\Delta^{-g}) = 0.567 - 0 - 0.367 = 0.2$

Because the information gain of concept g is highest, this concept will be queried first. In case g is false, concept d or e must be queried. Because $Gain(\Delta^{-g}, e)$ is greater than $Gain(\Delta^{-g}, d)$, concept e is queried first. Ordering the concepts in this manner, the decision tree as depicted in Figure

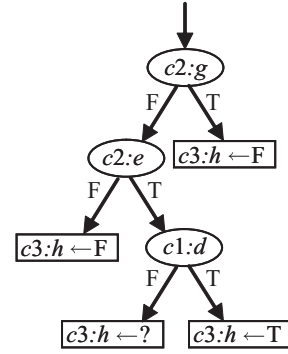


Figure 2: Decision tree for Ag-4

2 is obtained. In this figure, a circle around a concept means that its truth-value must be found out; a box around a concept means that the truth-value for the information need can be derived.

We have now discussed in which order information must be obtained. In the next section, we will discuss by which means information can best be obtained.

3.3 Query vs Subscribe

By querying the truth value of a concept c , the agent gets a response regardless of whether c is true or c is false. Besides querying, a common interaction mechanism in ubiquitous computing and peer-to-peer systems is the Publish-Subscribe protocol [14]. By subscribing to a concept c , the agent requests to be notified whenever c is true. This reduces the communication flow as information is only exchanged when c is true. When no notification is received, the agent can derive that $\neg c$ is the case. In this way, subscriptions introduce a local closed-world assumption in an open-world knowledge base.

As an intuitive example of the subscribe mechanism, consider a fire-department. Instead of repeatedly calling the residents of its district to ask if help is needed, the fire department requests to be notified about this.

In order to decide whether to query or to subscribe to a concept, it does not suffice to estimate the resulting communication load for the current time point only. Because subscriptions usually pay off after a number of time steps, an agent should stick to one plan which is expected to be most efficient for all future communications.

The costs of a plan indicate the expected number of sent messages per time step that result from carrying out the plan. This is calculated as follows. A plan is represented using labels which cover the nodes in the decision tree. If a node in the decision tree is covered, it means that the agent has sufficient information to traverse through the tree at that node. A label is either a Subscribe-label or a Query-label and is introduced before it is used to cover nodes in the tree. The two types of labels differ with respect to the costs that are associated with introducing the label, and with respect to the amount of nodes they can cover after they are introduced. The costs of a label corresponds to the average amount of sent messages that are caused by the introduction of the label. An agent continues with introducing labels until each (non-leaf) node in the decision tree is covered. The

costs associated with this labelling equals the total amount of costs of introducing the labels.

By specifying which nodes a query label can cover and what the associated costs are, Query-labels are characterized as follows:

DEFINITION 4. *Costs and covering of Query-labels*

- A label $QUERY(c)$ covers one node about concept c .
- The costs of a label $QUERY(c)$ is the probability that the node which it covers is reached.

Note that, when there are two nodes with c in the decision tree, the label $Query(c)$ has to be introduced two times such that costs of both labels are added to the total amount of costs. This is not the case for Subscribe labels, which are characterized as follows:

DEFINITION 5. *Costs and covering of Subscribe-labels*

- A label $SUBSCRIBE(c)$ covers all nodes about concept c .
- The costs of a label $SUBSCRIBE(c)$ is the probability that c is true.

Note that the above definition also applies to negated concepts, i.e. by calculating the costs of $SUBSCRIBE(c)$ and $SUBSCRIBE(\neg c)$, the choice can be made whether to get notified when c is true or when c is false.

Typically, the root node of the decision tree is a suitable candidate for subscription. Because it is always necessary for the agent to have information on this concept, the query-costs would be 1, whereas the subscription costs would be less than 1. The lower the node is situated in the tree, the less likely it becomes that information about that concept is needed and the less the query costs become. At the same time, subscription costs remain the same. Thus, the nodes at the bottom of the tree are typically labelled with query.

The ideas introduced in this section are illustrated in the following example:

EXAMPLE 3. *Consider Ag-4 in Figure 1 and the relevant ABox information from Example 2. The costs of different ways of obtaining information are listed below.*

- $Costs(QUERY(g)) = 1$
- $Costs(SUBSCRIBE(g)) = Pr(g) = \frac{9}{15} = 0.6$
- $Costs(SUBSCRIBE(\neg g)) = Pr(\neg g) = \frac{6}{15} = 0.4$
- $Costs(QUERY(e)) = Pr(\neg g) = \frac{6}{15} = 0.4$
- $Costs(SUBSCRIBE(e)) = Pr(e) = \frac{8}{15} = 0.533$
- $Costs(SUBSCRIBE(\neg e)) = Pr(\neg e) = \frac{7}{15} = 0.467$
- $Costs(QUERY(d)) = Pr(\neg g, e) = \frac{5}{15} = 0.333$
- $Costs(SUBSCRIBE(d)) = Pr(d) = \frac{5}{15} = 0.333$
- $Costs(SUBSCRIBE(\neg d)) = Pr(\neg d) = \frac{10}{15} = 0.667$

Using this information, the optimal labelling becomes as depicted in Figure 3

In this section we have focussed on how information can be efficiently obtained from one source. When information must be obtained from multiple sources, the mutual relation between these sources can be used to further reduce the communication load, which is the topic of the next section.

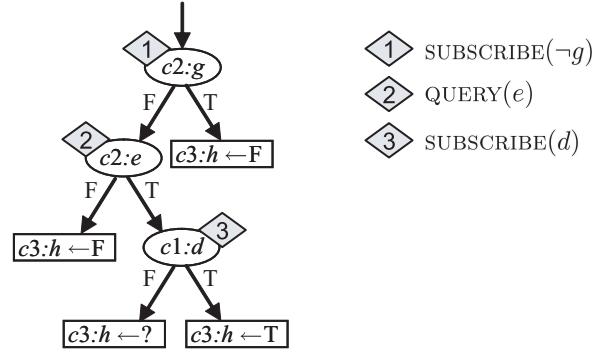


Figure 3: Labelled decision tree for Ag-4

3.4 Conditional Subscriptions

By using a conditional subscription, an agent requests to receive notifications of something only if some condition holds. This can be useful when the data that must be combined to come to a conclusion is not probabilistically independent. For example, consider Ag-1, Ag-2 and Ag-5 in Figure 1 and suppose that Ag-5 has information need i (*rainbow*). To satisfy its information need, Ag-5 must know the truth values of c (*rainy*) and d (*sunny*). *rainy* is frequently true and *sunny* is frequently true, but they are very rarely true at the same time. If Ag-5 would subscribe with Ag-1 for c and with Ag-2 for d , it would receive a lot of notifications. By using a conditional subscription on c if d (written $(c|d)$), Ag-5 requests Ag-1 to send notifications about c only when d is also true. Of course, this requires Ag-1 to have information on d which it can obtain from Ag-2. The redirection of the information flow is depicted in the following figure.

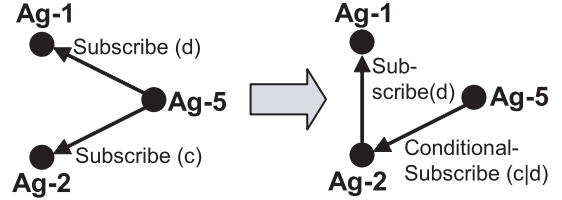


Figure 4: Redirecting the information flow

To judge whether conditional subscriptions actually lead to a reduction of the overall communication load, we must calculate the costs of conditional-subscribe labels and state precisely which nodes these labels can cover. An agent that is conditionally subscribed to $(c|d)$ receives a notification when c and d are both true. Therefore, this label covers both nodes c and d .

To calculate the costs of a conditional subscription on $(c|d)$, the subscribing agent (Ag-5 in Figure 4) should not only compute the costs of receiving notifications on c and d (the arrow between Ag-5 and Ag-2), but also the costs of acquiring information about d it burdens the other agent with (the arrow between Ag-2 and Ag-1). An agent can estimate these costs by inquiring the other agent about it, or by computing what it would cost if it would have to subscribe to the concept itself. This is specified in the following definition:

DEFINITION 6. *Costs and covering of Conditional-subscribe labels*

- A label $\text{CONDITIONAL-SUBSCRIBE}(c|d)$ covers the nodes about c and d .
- The costs of a label $\text{CONDITIONAL-SUBSCRIBE}(c|d)$ is equal to the probability that c and d are true plus the estimated minimal costs of acquiring information on d .

An illustrating example is given below.

EXAMPLE 4. *Consider Ag-5 in Figure 1 with information need i and the relevant ABox information from Example 2. The costs of different ways of obtaining information is:*

- $\text{Costs}(\text{CONDITIONAL-SUBSCRIBE}(d|c)) = \text{Pr}(d \sqcap c) + \text{Pr}(\neg c) = \frac{1}{15} + \frac{4}{15} = 0.33$ (Note that Ag-5 has estimated the minimal costs of obtaining information on c as $\frac{4}{15}$, i.e. the costs of subscribing on $\neg c$)
- $\text{Costs}(\text{SUBSCRIBE}(\neg c)) = \text{Pr}(\neg c) = \frac{4}{15} = 0.267$
- $\text{Costs}(\text{SUBSCRIBE}(d)) = \text{Pr}(d) = \frac{5}{15} = 0.333$
- $\text{Costs}(\text{QUERY}(d)) = \text{Pr}(c) = \frac{11}{15} = 0.733$

This reveals that the plan which uses a conditional subscription on c and d only costs 0.33. A plan which uses a subscription on $\neg c$ and a subscription on d would be more costly, i.e. $0.267 + 0.333 = 0.6$. Hence, the optimally labelled decision tree becomes as follows:

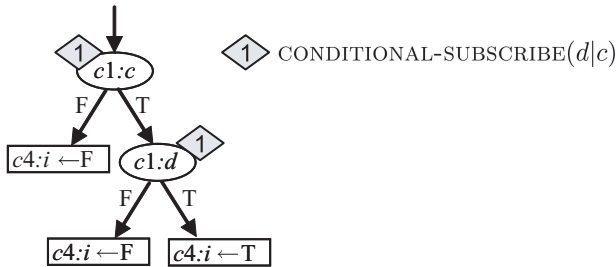


Figure 5: Conditional subscriptions

Note that, in case the agent does not receive a notification when it is conditionally subscribed to $(c|d)$, it does not know which of the two concepts c or d is actually false. Therefore, it does not know whether to follow the left edge below node c or the left edge below node d (Figure 5). In this case, the choice is irrelevant, because both edges lead to the same result. However, if the subtree under the left edge of c is different than the subtree under the left edge of d , problems can arise. Therefore, we restrict the use of conditional subscriptions on $(c|d)$ to those situations where $\neg c$ and $\neg d$ have the same consequences.

4. EXPERIMENTAL VALIDATION

We have implemented a test environment, called Ubismart, which allows developers to easily prototype a ubiquitous computing system corresponding to the architecture discussed in this paper. In this way, hands-on experience can be obtained with the design of these types of systems. Furthermore, simulation experiments can be performed to study the information flow between the different components. We will first describe the Ubismart environment in more detail. Then we will discuss the implementation of Ubismart and

argue how our conceptual framework introduced in the previous sections translates to technology. Finally, we will discuss the experiments we have performed to test the different communication optimization techniques that are introduced in this paper.

4.1 Ubismart

A Ubismart experiment starts with an empty model which can be populated with agents of different types. Following [3], we distinguish between Sensors, Interpreters and PDA's which characterize different roles an agent can play in a system. A sensor obtains information by sensing its environment, an interpreter acquires information from one or more sensors and derives a higher-level interpretation from this and a PDA presents information to the user. Figure 6 shows a screenshot of a Ubismart experiment. The camera icon represents a sensor, the computer icon represents an interpreter and the PDA icon a PDA.

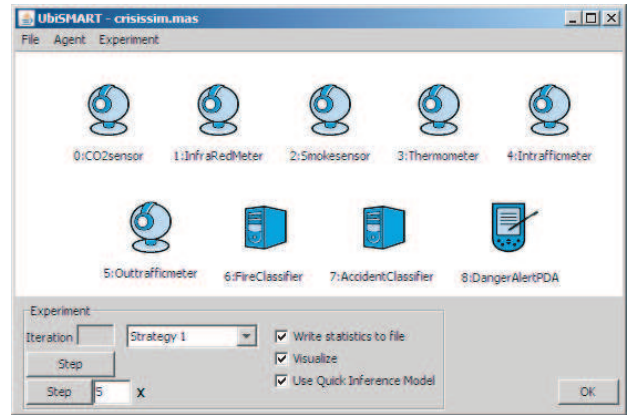


Figure 6: Screen shot of Ubismart experiment

By clicking on an icon, a window is opened which can be used to configure the agent. Most importantly, the ontology of the agent is selected here. For sensors, also the location of its sensory input must be specified. For PDA's, also the information needs must be specified, i.e. which information the PDA is supposed to present to the user.

4.2 Building on existing technology

Ubismart is programmed in Java and is built upon a number of technologies which are currently receiving a lot of attention in the AI community. In this section, we will briefly discuss each of them and the role they play within Ubismart.

The ontologies are specified in OWL-DL which is a specific species of the OWL language [15], a language for specifying ontologies developed by the semantic web community. OWL contains a number of nice features which make it particularly suitable for our purposes. Firstly, it has a formal semantics that corresponds to description logic, which is the formalism we have used in the specification of our approach. Secondly, it allows one ontology to import another ontology which is useful for implementing layered ontologies. For example, the layered ontology of Ag-4 would be constructed by importing the ontologies $c1$, $c2$ and $c3$ and specifying additional statements for the mapping between them. Thirdly, as the language is based on XML, it contains namespaces, i.e. unique identifiers which are used as prefixes of concept names to avoid name clashes. In our implementation, namespaces are

useful to specify the context in which a concept is defined.

Whereas OWL nicely conforms to all kinds of syntactic standards, the language is not very well readable for humans. Therefore, we have used Protégé [9], which is a graphical ontology editor containing an open source Java library for OWL. To understand complex ubiquitous systems involving multiple ontologies, proper visualization of ontologies is crucial. We have applied the Protégé-OWL API to deal with OWL models in Java and to graphically represent the ontologies involved in the user interface.

The agents must also be capable of performing ontological reasoning, for example to derive the list of informative concepts (Definition 1). We use FaCT++ [18] as a description logic reasoner. Because description logics are designed to have nice computational properties, all reasoning in Ubismart proceeds in a timely fashion.

The data which enters the system through the sensors is modelled using a Bayesian network. In this way, we can specify the probability measure of the sensor inputs as well as the probabilistic dependencies between them. We have used Hugin [12] as a tool to both create the Bayesian network as well as to generate data that conforms to this network.

4.3 Experiments

The research reported in this paper is performed as part of the ICIS project [1], a large national research project on distributed information systems for crisis management. The experiments we describe in this section are tailored to the usage scenario of this project, i.e. effective computer assistance for a major traffic accident in a tunnel. A modern tunnel contains a wealth of sensors to monitor the composition of air, the presence of smoke, the number of vehicles in the tunnel, the average driving speed, etc. The Ubismart approach can help the crisis workers gain access these sources of information via their PDA's, which could help to improve their safety and efficiency.

In this paper, however, we do not intend to model the full system which is needed for effective crisis management. Rather, we will assume a simple set-up of sensors and computers which suffices to study those aspects we have discussed in this paper, i.e. the reduction of communication load by using decision trees, subscriptions and conditional subscriptions.

The system contains six sensors, two interpreters and one PDA (see Figure 6). Twelve different ontologies are involved, among which the ontologies used by interpreters that specify how sensor data can be used to derive higher forms of knowledge. For example, lower forms of information such as infrared (which indicates the presence of flames), smoke, temperature, and carbon dioxide are used to derive a potential fire. The amount of incoming traffic combined with the amount of outgoing traffic is used to derive a traffic jam inside the tunnel (which could be the sign of an accident). We modelled the sensor data in a way we believed to be realistic. For example, infrared level is almost always low; the smoke level gets high more often due to car exhausts; heavy traffic frequently occurs, and a dependency exists between the probability that heavy traffic is entering the tunnel and that heavy traffic is leaving the tunnel.

To demonstrate that the efficiency measures we have discussed in Section 3 actually reduce the information flow in the system, we define the following strategies:

- **st1**: Query all informative concepts in random order

(see Definition 1) until the information needs are satisfied.

- **st2**: Query all informative concepts ordered by information gain (see Definition 3)
- **st3**: Query or Subscribe to the informative concepts ordered by information gain taking into account the costs (see Definition 4, 5)
- **st4**: Query, Subscribe or Conditionally subscribe to the informative concepts ordered by information gain taking into account the costs (see Definition 4, 5, 6)

We have performed four experiments with each of the different strategies. Each experiment lasted for 1000 time steps. We measured the average amount of sent messages that was needed per time step before every component's information needs were satisfied. In order to enable the agents to form a reliable estimation of the costs of a subscription (in st3 and st4), agents only decided whether to query or to (conditionally) subscribe after a short training period (approximately 60 time steps).

We assumed that the probability measure of the incoming sensor data remained equal throughout the whole experiment. However, the technique can be easily extended to a scenario where the probability measures change such that an agent must repeatedly reassess its decision which interaction mechanism to follow.

4.4 Results

The results of the experiment are shown in Figure 7.

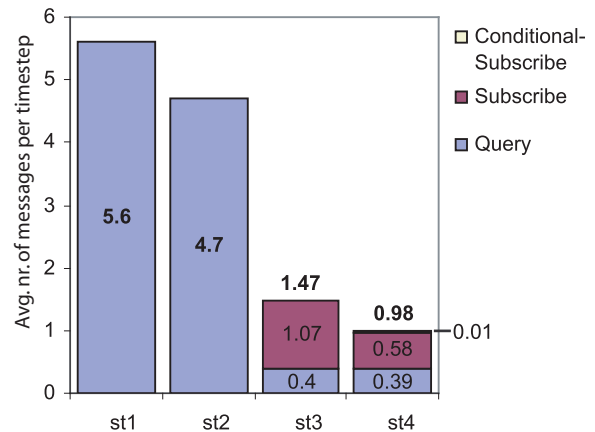


Figure 7: Results

As appears from this graph, every efficiency measure we have introduced in Section 3 leads to a significant reduction of communication load in the system. The overall reduction in communication load (comparing st1 to st4) is over 80%. Obviously all communication in st1 and st2 was due to Queries. In st3, more than two third of the communications was caused by the subscribe mechanism. In st4, the conditional subscribe mechanism, which formed 1% of the total communication flow, replaced a significant part of the communications of the subscribe mechanism.

Of course, the quantitative results are dependent on the particular ontologies used in the system and on the Bayesian networks which feed the sensors. Hence, we cannot say that for *all* ubiquitous systems the reduction in communication load will be 80%. However, the amount of messages required to satisfy the agents' information needs will never be

more using st4 than using st1. This is because the communications that are generated by st1 are also allowed by st4 and the agents only apply the more advanced communication techniques if this leads to increased efficiency. In most systems however, the sensor data has a certain underlying probability distribution and not all combinations of sensory inputs are relevant for each agent's information needs. In these systems, the techniques discussed in this paper are successful.

5. CONCLUSION

Over the last two decades, the field of MAS has given birth to many applied research areas. Not only has the MAS paradigm shaped the landscape of these applied research fields, also have these applied research areas frequently returned valuable insights for the MAS paradigm as a whole. In this way, we believe that our research on communication in ubiquitous computing is a valuable contribution to the field of communication in MAS's, in particular for those domains where information is abundant but where information exchange is not free of costs.

In the introduction we have described the ultimate goal of achieving serendipitous interoperability between ubiquitous computing devices. Of course, we cannot claim to have fully achieved this goal. Whether two agents understand each other depends on whether their developers have independently decided to use the same parts of the ontology. It is expected that, when building software with ontologies becomes common practice, every domain will have its own standard and widely used ontologies. Agent developers will be eager to equip their agents with a popular ontology to make them interoperable with as many other agents as possible. Of course, this remains to be verified by practice.

Nevertheless, our architecture remains useful also when a less ambitious form of interoperability is assumed and some prior alignment of the ontologies has been performed by the system developers. Although some central coordination of the ontology-types of the agents is assumed in this scenario, we still do not make any demands about which components are present at run-time.

The main challenge we faced in this paper was that of reducing the communication load between agents. Whereas this issue has been frequently addressed from a network-level perspective, we have taken a semantic perspective. We have shown how the information from the agent's ontologies and ABoxes can be used to reduce the communication load. Our experiments have revealed that the benefits may be considerable. In our opinion, these benefits outweigh the extra implementation efforts in many MAS applications, in particular ubiquitous computing applications.

In the future, we intend to perform experiments in a virtual environment to study the effects of our approach on human decision making in crisis management.

6. REFERENCES

- [1] www.icis.decis.nl/.
- [2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The description logic handbook: Theory, implementation and applications*. Cambridge University Press, 2003.
- [3] A. Dey, D. Salber, and G. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16, 2001.
- [4] T. Finin, R. Fritzon, D. McKay, and R. McEntire. KQML as an agent communication language. In *Proceedings of CIKM*, pages 456–463. ACM, 1994.
- [5] FIPA. FIPA Communicative Act Library Specification. <http://www.fipa.org/specs/fipa00037/>, 2002.
- [6] E. Giunchiglia, P. Traverso, and F. Giunchiglia. Multi-context systems as a specification framework for complex reasoning systems. *Formal Specification of Complex Reasoning Systems*, pages 45–72, 1993.
- [7] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.
- [8] S. Gurun, P. Nagpurkar, and B. Y. Zhao. Energy consumption and conservation in mobile peer-to-peer systems. In *MobiShare*, pages 18–23. ACM, 2006.
- [9] H. Knublauch, R. Ferguson, N. Noy, and M. Musen. The protégé OWL plugin: An open development environment for semantic web applications. In *3rd Int. Semantic Web Conference*. Springer, 2004.
- [10] O. Lassila. Serendipitous interoperability. In E. Hyvönen, editor, *The Semantic Web Kick-Off in Finland - Vision, Technologies, Research, and Applications*. HIIT Publications, 2002.
- [11] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.
- [12] A. Madsen, F. Jensen, U. Kjærulff, and M. Lang. HUGIN -the tool for bayesian networks and influence diagrams. *International Journal of Artificial Intelligence Tools*, 14(3):507–543, 2005.
- [13] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [14] A. Ranganathan and R. H. Campbell. An infrastructure for context-awareness based on first order logic. *Personal Ubiquitous Computing*, 7(6):353–364, 2003.
- [15] M. Smith, C. Welty, and D. McGuinness. OWL Web Ontology Language Guide. <http://www.w3.org/TR/owl-guide/>.
- [16] M. Stefik. *Introduction to knowledge systems*. Morgan Kaufmann Publishers, 1995.
- [17] P. Stone and G. Weiss. Editorial. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*. ACM, 2006.
- [18] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *Proceedings of the IJCAR*, volume 4130 of *LNAI*, pages 292–297. Springer, 2006.
- [19] J. van Diggelen, R. J. Beun, F. Dignum, R. M. van Eijk, and J. J. C. Meyer. Ontology negotiation: Goals, requirements and implementation. *International Journal of Agent-Oriented Software Engineering (IJAOSE)*, 2007.
- [20] J. van Diggelen, R. J. Beun, R. M. van Eijk, and P. J. Werkhoven. Modeling decentralized information flow in ambient environments. In *Proceedings of ambient intelligence developments (AmI.D '07)*, 2007.
- [21] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, 1991.