CONTRACT BASED COOPERATION FOR AMBIENT INTELLIGENCE: PROPOSING, ENTERING AND EXECUTING CONTRACTS AUTONOMOUSLY

by

Fatma Başak Aydemir

B.S., in Computer Engineering, Boğaziçi University, 2009

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University
2011

ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to my thesis advisor Assoc. Prof. Pınar Yolum for her invaluable guidance, patience, encouraging support and being a role model. This thesis would not be possible without her efforts. My appreciations go to my jury members, Dr. Suzan Üsküdarlı and Assoc. Prof. Şule Gündüz Öğüdücü.

My friends in MAS Research Group, Akın Günay, Özgür Kafalı, and Reyhan Aydoğan provided me their friendship and guidance. My friends in AILAB, Çetin Meriçli, Abuzer Yakaryılmaz, Tekin Meriçli, Barış Gökçe, Serhan Daniş and Haşim Sak provided me a fun working environment. Ayça Kundak Işıksal shared the most of this journey with me. I would like to thank them all.

I would like to thank the members of the executive board of the Institute for their guidance in general. I also would like to thank Nuran Marangoz, Semiha Mutlu and Süleyya Şen for their support and the warm working environment.

I would like to express my gratitude to my parents Nuray and M. Gıyasi Aydemir for being the perfect parents. I thank them for their self devotion and patience. I also would like to thank my little brother, Engin Aydemir, for bearing me throughout his life. I would like to thank Eren Özek for his support, patience, and love.

Parts of this thesis is accepted to be published in the proceedings of STAMI 2011 and EASSS 2011 Student Session. This thesis has been supported by Bogazici University Research Fund under grant BAP5694 and by the Scientific and Technological Research Council of Turkey (TÜBİTAK) 2210 National Graduate Scholarship Program.

ABSTRACT

CONTRACT BASED COOPERATION FOR AMBIENT INTELLIGENCE: PROPOSING, ENTERING AND EXECUTING CONTRACTS AUTONOMOUSLY

As we are introduced with the more complex, high technology devices, the need to smoothly integrate these new devices to our lives grows. In order to satisfy this need, a new kind of pervasive and ubiquitous intelligence, Ambient Intelligence, is suggested. Ambient Intelligence (AmI) describes environments that sense and react to the humans in time to help improve their living quality. Software agents are thus important in realizing such environments. While existing work has focused on individual agent's reactions, more interesting applications will take place when agents cooperate to provide composed services to humans. When cooperation is required, the environment needs mechanisms that regulate agent's interactions but also respect their autonomy.

Accordingly, this thesis develops a contract-based approach for offering composed services. The system consists of two ontologies that keep knowledge about the environment and the domain, agents that represent the user, and the agents that provide services. Similar to the real life, individual services are usually offered by different agents. Agents that are going to take part in the contracts are selected according to their capabilities. At runtime, selected agents autonomously decide whether they enter contracts. The decision is based on the knowledge which is described in the ontologies and the internal state of the agent. After the agents agree on the contracts, they then act to fulfill their contracts. We apply this multiagent system on an intelligent kitchen domain and show how commitments can be used to realize cooperation. We study our application on realistic scenarios.

ÖZET

ORTAM ZEKASI İÇİN SÖZLEŞME TABANLI İŞ BİRLİĞİ: BAĞIMSIZ OLARAK SÖZLEŞME ÖNERME, KABUL ETME VE KOŞMA

Hayatımıza yüksek teknoloji ürünü karmaşık cihazlar girdikçe, bu yeni cihazların sorunsuzca ve hissettirilmeden günlük yaşantımıza tümlenmesi ihtiyacı ortaya çıkar. Bu ihtiyacın giderilmesi için aynı anda her yere yayılmış, ortam zekası denilen yeni bir zeka türü ortaya atılmıştır. Ortam zekası, insanların hayat kalitesini yükseltmek için insan varlığına duyarlı ve tepki veren ortamları anlatır. Yazılım etmenleri, bu ortamları hayat geçirmek için önemlidir. Bugüne kadar yapılan araştırmalar bireysel etmenlerin tepkilerine yoğunlaşmışken, etmenlerin insanlara bileşik servisler sunmak için iş birliği yaptıklarında daha ilginç uygulamalar ortaya çıkacaktır. İş birliği kaçınılmaz olduğunda, ortamın etmenler arasında etkileşimi düzenlerken etmenlerin özerkliklerine saygı duyacak düzeneklere gereksinimi vardır.

Bu çalışma bileşik servislerin sağlanmasında sözleşme tabanlı bir yaklaşım sunar. Sistem; alan ve çevre bilgisini saklayan iki ontoloji, kullanıcıları temsil eden etmenler ve servis sunan etmenlerden oluşur. Gerçek hayatta olduğu gibi, bileşik servisi oluşturan bireysel servisler çoğu zaman farklı etmenlerce sunulur. Sözleşmelerde yer alacak etmenler, sağladıkları servislere göre seçilir. Yürütüm aşamasında seçilen etmenler sözleşmelere girip girmeme konususnda özerk biçimde karar verirler. Etmenlerin bu kararı ontolojiler üzerinden mantık yürüterek elde ettikleri sonucu kendi iç durumlarıyla birleştirmeleriyle ortaya çıkar. Sözleşmeler üzerinde anlaşma sağlandıktan sonra etmenler sözleşmelerin gereklerini yerine getirmeye çalışırlar. Bu çok etmenli sistemi akıllı mutfak alanına uygulayıp iş birliği için sözleşmelerde nasıl faydalanılabileceğini gösteriyoruz. Uygulamamız için gerçekçi senaryolar kullanıyoruz.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

ACL	Agent Communication Language
AmI	Ambient Intelligence
BDI	Belief–Desire–Intention
Cobra	Context Broker Architecture
CMA	Coffee Machine Agent
DAML	DARPA Agent Markup Language
FA	Fridge Agent
FIPA	The Foundation for Intelligent Physical Agents
FOAF	Friend of a Friend
JACK	Java Agent Component Framework
JADE	Java Agent Development Framework
JASON	Java Agent Speak Interpreter
MA	Mixer Agent
MAS	Multiagent System
OSGI	Open Services Gateway Initiative
OWL	Web Ontology Language
RDF	Resource Description Framework
SOAP	Simple Object Access Protocol
UA	User Agent

1. INTRODUCTION

Development of technological devices that are small enough to integrate into the human life smoothly has lead to a new intelligence called Ambient Intelligence (AmI) [1, 2]. AmI indicates environments that are aware of and responsive to human presence. AmI should be distributed over the environment and it should be transparent.

The pervasive and ubiquitous characteristics of AmI can be realized by multiple components' working together. Many integral parts of AmI, from sensors to implanted micro chips, are embedded into the environment. Software or hardware agents, home devices and other parts of the system collaborate to serve the user of the system. Sensors provide data which is processed by the software agents. Home devices act according to the conclusions driven by the processed data.

The intelligence, which is spread over the environment, should be personalized so that it can satisfy the needs of different users. One size fits all is not an applicable feature for AmI. Users of the system will most likely have unique desires and needs. Even the system is used by one user, the factory settings may not work for her. Therefore, the system should be customizable according to each user's preferences.

The preferences of the user may vary for different situations. AmI should be aware of the environmental changes. The weather forecast outside the house, the humidity level, the temperature in the house may be significant for different scenarios. The context of the user is important as well. The change in the medical conditions of the user, the heart rate, blood pressure or a new disability should have an effect on how the system serves the user. So, AmI should be aware of both the environment's and the user's context.

AmI has emerged to be applied on the humans' daily life. Change of situations is inevitable in human life. Environmental context changes frequently. The preferences

of the user may be changed throughout the time even for the same context. The system should not neglect the changes happening. It should adapt itself to the changes and act accordingly.

Although adapting to the changes is an important characteristics, in order to serve the user seamlessly, anticipating these changes and then adapting to them is also as significant. By anticipating the possible changes, the system can prepare to adapt them while serving regularly. Thus, the user do not notice the transition at all or it notices the difference very little [3, 4].

User feedback is one of the main performance measures for AmI systems. Learning from user feedback helps improving the quality of the AmI systems. It makes it easier to understand the preferences of the user when the user is not willing to set all of her preferences by herself. Learning mechanisms are also useful in detecting changes in user preferences. Thus, such systems accelerate the adapting process.

AmI can cover very large areas, or it may cover small spaces in great detail. For such systems, many components need to work together. Therefore, the scalability of the system has great impact on the success of the system. AmI systems should continue to work properly when many components are attached to it. So far, we have talked about how dynamic AmI environments are. Such dynamism also exists for the existence of the components. Components of the system may leave the system, or new components may be added to the system in an ad hoc fashion. Also, there may be some unanticipated situations. The system should be flexible enough to handle such situations.

In an AmI system, communication among components has a vital role in satisfying the user's needs. Following some standards for this communication ensures that the intended meaning of the sender's message is the same as the meaning understood by the receiver. Communication standards also secures the continuity of the communication when the communication structure between agents is changed. Communication standards such as FIPA standards also enables inter communication

between different AmI systems. The FIPA standard is one of the communication standards that follow the speech act theory, which states that the communication is a form of action [5]. In FIPA standards, each message is labeled according to the intent of the sender agent.

Since AmI is pervasive, it is important to convey the gathered knowledge to the system components that need to use this information. One problem is to find the components that are in need of the information. If the devices that need the information are not carefully identified, all components will be overwhelmed with irrelevant data. Using ontologies for knowledge representation solves this problem. When the knowledge is represented in ontologies, components in need of information can check the ontology themselves.

Bottleneck designs are not suitable for AmI since a failure in the system may lead the whole system to lose its working capability, therefore causing the environment to lose the intelligence. So, centralized designs should be avoided. Components should be distributed, so that a fault in a component should not affect the other components' performance.

Some emerging technologies for AmI include simple sensors, radio frequency identification, affective computing, brain computer interfaces, nanotechnology, sensors that detect sensors and mobile sensors. Basically, all kind of sensors are used to monitor data, these are usually not smart devices. Radio frequency identification has unobtrusive characteristics and unobtrusive features, which makes them useful for the AmI applications. Affective computing and brain computer interfaces are utilized to integrate human thinking and emotions more to AmI [6].

Some other supporting technologies for AmI are energy supplies, smart materials, networking and communication, grid computing, peer to peer network architectures, context aware software and systems, and new shapes of computational devices. High technology energy supplies provides green and efficient ways of supplying energy. Networking and communication, and also peer-to-peer networking are inseparable

from AmI since there are many server client architectures and networks in an AmI environment such as body area network and personal area network. Grid computing enables controlling resources efficiently. Smart materials and new shapes of computational devices bring intelligence to mundane objects of human life. Context aware software has significant role in personalization, adaptation and anticipation [6].

Another important emerging technology for AmI is software agents. Software agents serve for AmI systems in various ways. Autonomous, intelligent agents process data from the sensors, reason on these data, learn from the data, make plans and take actions according to the data. In order to develop robust and scalable software systems, autonomous agents, that can engage in high level social interactions, operate within flexible organizational structures and manage to reach their objectives in a dynamic environment, are required [7]. Considering the complexity of an AmI system, multiple agents are needed to collaborate to serve properly. With this consideration, AmI can be realized by multiagent systems.

Multiagent systems (MAS) consists of several intelligent agents that communicate. Multiagent systems may be open or closed depending on the policy of the system regarding agents' leaving and entering the system. In closed systems, agents cannot leave or enter the system. On the other hand, in open systems agents may leave or enter the system during computation [8]. Open multiagent systems are favorable for AmI applications, reflecting the dynamism of AmI systems.

In MAS, each agent has some degree of autonomy with respect to their actions. Agents autonomously work and communicate with each other when necessary. Overall capability of the MAS may exceed the capabilities of the individual agents [8, 9]. System tasks may be divided into smaller pieces, enabling autonomous agents to make their own contribution to the system task.

There are different strategies for agents to work together. Two of these strategies are cooperation and collaboration. In cooperation, agents' authority is the highest, whereas in the collaborative work agents more act upon a defined collaborative struc-

ture. Cooperative strategies last for a short time; however the collaborative strategies last longer [10].

Communication among agents is one of the key elements that enable them to work together. Agents may communicate via messages. The messages they transfer should have a meaning and also a binding power on the agents. Following speech act theory, we may conclude that a message from one agent to another reflects the intended action of the agent that sends the message [5]. Standards such as FIPA, labels the messages so that they carry the intent of the sender agent.

Commitments form a contractual relationship between the agents [11]. An agent may have commitments to different agents to take certain actions. These type of commitments are social commitments. Social commitments are contracts made from one agent to another in order to satisfy certain properties [12,13]. Commitments regulate the interaction among agents. They are independent of the underlying messaging structure.

In this thesis, we propose a cooperative AmI system which consists of autonomous agents. Agents are not forced to serve or take any actions that they do not intend to do so. Cooperation between agents are also up to the agent themselves, there is not a collaboration structure set between agents.

The system is dynamic in various ways: resources can be added or consumed, agents may enter and leave the system or they can change the services they provide. We follow a user centered design focusing on the user's needs and demands [14] for this open MAS, as it is consistent with the human-centric nature of the AmI systems. One of the intelligent agents represents the user of the system and it is called User Agent (UA). UA tries to satisfy the needs and desires of the user. It operates like a bridge between the user and the AmI. It frees the user from the task of dealing with the complicated and detailed technical structure and the communication network of the system. Other agents, which have different skills, cooperate with UA in order to satisfy the user's needs. Most of the time, a single agent is not capable of serving

all the needs and desires of the user by itself. Our system uses the power of MAS in order to combine various services from various agents to serve the user the composed services of her choice.

One distinguishing aspect is that predefined contracts, which are generated before agent interaction do not exist in the system, because such static structures do not apply well to the dynamism of the system described above. Instead of relying on predefined contracts, relevant contracts are created in conformity with the internal states of the parties during agent interactions. This brings into the picture the second important characteristics of our system, that each agent exercises its autonomy in entering contracts. To decide on entering or proposing a contract, each agent models its internal state in detail using an ontology. The ontology is also used by an agent to find out the requirements of a task that it can commit to. Thus, an agent can reason whether it can actually perform a task based on the requirements of the task and its current internal state. The internal states of the agents are not visible to other agents and the agents decide whether or not to take part in the contracts themselves.

As we opt for a flexible system, UA seeks for other agents to interact and tries to establish other contracts that guarantees realization of the properties needed to satisfy the user when a contract cannot be created. UA may also try to satisfy the other agents first, in order to pursue them to serve what it needs to get in order to satisfy the user.

One of the main contributions of this thesis is that we do not force our agents to participate in rigid collaboration structures that are set in compile time. Instead of participating in such rigid collaboration structures, our agents form flexible cooperative structures in run time. We utilize contracts that are generated dynamically in run time to form such cooperative structures.

Another significant contribution of this work is that agents make the decision of entering the contracts themselves by considering the state of the system, their internal state and the knowledge represented in the ontologies that is accessible and

relevant to the situation. Agents reason on the ontologies in order to decide whether they can serve or not as opposed to how they should serve. The system does not force the agents to serve or we do not unrealistically assume that the agents can serve all the time.

The rest of the thesis is organized as follows: Chapter 2 explains the benefits of the dynamically generated contracts over the statically generated ones. Our arguments are supported with example scenarios. Chapter 3 describes our approach to design such MAS. The architecture of the system, architecture of the agents, components of the agents and the algorithms developed are given in detail. In Chapter 4 technical details, technologies used and the issues faced with are specified. Chapter 5 illustrates selected scenarios and the execution results of these scenarios on our example domain. The scenarios are constructed in order to cover a broad range of situations from the simplest to more complex ones. The flow of communication is illustrated and the steps of the computation is given in detail. Chapter 6 surveys some AmI applications that involves agents. Such applications are discussed in terms of key features of AmI and the results are summarized at the end of the chapter. This chapter also includes the discussion of our work, possible improvements, future work and our final remarks.

2. GENERATING CONTRACTS

A contract between agents X and Y is represented as CC(X,Y,Q,P) and interpreted as the debtor agent X is committed to bring the proposition P to the creditor agent Y when the condition Q is realized. Contracts assure that the creditor obtains the promised properties and ease the process of tracing the source of possible exceptions [11].

2.1. Statically Generated Contracts

In some multiagent systems certain characteristics of the system are static. For example, the static structure of the system may appear in the roles of agents in the system. Roles of the agents in the system may not change during computation, i.e an agent with a specific role acts according to its role throughout the execution or capabilities of an agent, i.e. the services an agent is able to serve may be fixed. The types of the resources to realize the capabilities of the agents may be unchanging. The amount of resources that is reachable by the agents may be unlimited.

Consider a multiagent AmI system with UA and two other agents, Agent 1 and Agent 2. UA always tries to serve only a single type of service bundle since the user does not change her preferences about the service bundles once she announces them. Agent 1 is able to serve Service 1 from the bundle, whereas Agent 2 can serve Service 2. Agents 1 and 2 need some resources to serve their services, however in this case, they never run out of their resources. UA tries to assure that it gets Service 1 and Service 2 from other two agents whenever it makes a request.

Two statically generated contracts can be used to realize this scenario. Assume that the following contracts are generated at design time and adopted by the agents:

- CC(Agent 1, UA , Service 1 Request, Service 1)
- CC(Agent 2, UA , Service 2 Request, Service 2)

These two contracts specify that if UA makes a service request for the Service 1, Agent 1 is committed to bring out Service 1. Similarly, if UA makes a request for Service 2 then Agent 2 is committed to serve Service 2. In such static environments, statically generated contracts may be enough for the system in order to work correctly. Contracts may be specified during compile time, and the agents may follow the contracts at the run time. Since the system is not going to change at the run time, there is no reason to attempt to generate the contracts at run time. The system will work without a problem as long as the system keeps its static state.

2.2. Dynamically Generated Contracts

If the system is dynamic, then the statically generated contracts will not suffice. The scenario depicted above is far from being realistic. Any change in the environment breaks the static state of the system. The contracts that are created during compile time fails to cover the results of the changes that take place during run time. So, the system fails to satisfy the needs and desires of the user. Next, we consider why the dynamism is necessary.

2.2.1. Scarce Resources

Consider the case that the resources necessary to provide the services 1 and 2 are not always available any more. For example, Agent 1 may run out of Resource 1 that is fundamental to serve Service 1. So, Agent 1 fails to serve Service 1 when requested, although it is committed to serve it. Since Service 1 is a part of the bundle requested by the user, the user's desires cannot be satisfied by the system. For an AmI system, an unsatisfied user indicates the low performance of the system.

In such cases, the statically generated contracts described in Section 2.1 are not sufficient to realize the user's desires. Instead of following the statically generated contracts, the agents should be aware of the changes in the environment and their internal states. They should not agree on the contracts, when they know that they are not able to satisfy the propositions of the contracts, for sure. For this scenario,

Agent 1 that is out of Resource 1 may request Resource 1 from UA , in order to serve Service 1. Such a request includes a contract as following:

- CC(Agent 1, UA , Resource 1, Service 1)

This contract can be translated as when Resource 1 is provided, Agent 1 is committed to serve Service 1. It is then UA 's decision whether accept or reject this request. UA may choose to provide Resource 1 in order to get Service 1. Generation of this second contract, saves the system from failure. The user can be served with her bundle and be satisfied. The system's performance is improved. In order to realize this, we need to allow agents to decide to enter contracts and more importantly enable them to generate the contracts that they want to enter.

2.2.2. Open Environment

In an open environment, agents may leave the system, the agents that have left the system may come back, or new agents may enter the system. Agents may get damaged and be unable to serve. Previously damaged agents may get fixed and start serving again. In more realistic scenarios, not only agents but also some communication channels may be out of order. Agents that cannot communicate with other cannot cooperate with them.

When UA tries to serve a bundle of services, the states of the provider agents should be taken into account. It is not rational to expect an agent, that is damaged and unable to serve, to deliver the services it is committed to serve. Similarly, if an agent cannot be reached through the communication channels, since the service requests cannot be conveyed to this agent, it will be unaware that it should bring out its services.

So, UA should carefully select the agent with which it is going to interact when it tries to serve a service bundle to the user. For statically generated contracts, the agents are chosen before the agent interaction starts, so the current states of the

agents and the system are not considered. Obviously, statically generated contract are not as powerful as the dynamically generated ones in situations explained above.

Consider the scenario where Agent 1 leaves the system. Agent 1 can leave the system because the system is open. Assume that somehow the system anticipates Agent 1's leaving, therefore it arranges that a new agent called Agent 3 enters the system. Agent 3 is capable of serving the same services as Agent 1, including the Service 1, which is a part of the service bundle that is going to be served to the user. The statically generated contracts as in Section 2.1 will not be affective to get the bundle prepared since there is not a contract between UA and agent whose current state allows it to serve. In this case, UA should try to generate a new contract with an agent that can serve the service requested. A new contract with Agent 3 should be as follows:

- CC(Agent 3, UA , Service 1 Request, Service 1)

This new contract solves the problem of UA , since it is made with an agent that can satisfy the propositions in the current state.

2.2.3. Changing Services

A multiagent system does not necessarily contain agents that have fixed services. Agents may start to provide new services or stop providing some of the existing ones. Consider a television agent which have two subscriptions: news and sports. If a new movie subscription is added, the number of services that is given by this television increases. Similarly consider a coffee machine which can serve espresso and cappuccino. When the coffee machine cannot steam the milk any more, it also cannot serve cappuccino. So, it loses one of its services.

In such cases, making prior arrangements for a service bundle may not work due to the change of services provided by agents. One problem is that a service required for the bundle may not be given by any of the agents in the compile time. UA may

consider itself as failed, although it is possible to establish a contract including this service when the bundle is going to be served to the user in the run time. Another problem is that an agent may stop serving a kind of service that is in the contract generated in the compile time. Waiting for this service from that agent causes a waste of time, UA should look for alternatives to serve it.

Assume that the user's desires has changed and it asks for a bundle including Service 1 and Service 3. In the compile time, there are no agents that can serve Service 3, so UA cannot establishes any contracts including Service 3. However, Agent 1 gets modified and becomes capable of serving Service 3 in the run time. When the user requests her service bundle to be served, since it has no contracts established for a service that resides in the bundle, UA responds as it is not possible. However, it should be the case that when it is come to serve the user, UA should check the system, in order to find agents that can serve services from the bundle. When it discovers that Agent 1 is now capable of serving Service 3, the following contract is generated:

- CC(Agent 1, UA , Service 3 Request, Service 3)

This contract binds Agent 1 to serve Service 3 when it is requested by UA . Then the user is served with her desired bundle and gets satisfied.

3. APPROACH

We develop a multiagent ambient intelligence system that is based on contracts between agents. The main aim of our system is to reflect the basic characteristics of AmI discussed in Chapter 1. We propose a system that serves its user unobtrusively; that is, we do not want to overwhelm the user to instruct the system step by step. Instead, we take the desires and needs of the user and decompose the service bundle given by the user into individual services. We propose a high level system, and do not deal with the low level technical details, the technical communication structure among agents, sensors and so on. Our aim is to build a high level application by combining existing methods and methodologies with our innovative approach.

Many AmI applications operate rigidly. For example, they rely on closed systems as in [15–18]. Agents in the system have a predefined collaboration structure [17, 19]. Each agent of the system has a specific task which does not change throughout the computation. The capabilities of the agents are limited to their initial capabilities. Software agents are used for learning, data processing, reasoning and so on. It is not assumed that the agents can serve directly to the user with their actuators. Instead, the task of serving user physically is left to devices, which are not intelligent, not context aware, cannot reason or make decisions themselves [15–20].

Contrary to this rigidity in existing systems, in our approach, we model each device as a software agent and enable it to carry out interactions and decisions flexibly. Our system also reflects the characteristics of multiagent systems. We propose an open heterogeneous system where agents are leave and enter the system freely. The agents in the system serve various services. Most of the time, the task is needed to be done consists of several subtasks. In the system, there may not be an agent which can fulfill all of the subtasks. The strength of MAS arises in such cases. The agents work together, combine their abilities to serve the user fully. In our system, the agents cooperate with each other. We do not set a collaboration protocol between agents. Agents decide themselves whether or not to take part in the cooperative work. Such

approach is useful in dynamic systems such as AmI, in which there is a constant change. Collaboration structure may fail to cover all possible situations that may arise during computation.

Contracts are used in multiagent system to specify and monitor obligations among agents [21, 22]. They ease the process of tracing exceptions. In our case, we use contracts to enable cooperation. If two agents need to cooperate, they reflect this in the contract they enter. Aggreing on contracts ensures that the requested services will be served by the debtor, when they are needed. Hence, our idea of cooperation is on–demand and regulated by mutually agreed contracts

Knowledge should be represented formally in order to allow components of a system to easily access, update and share it. In order to represent knowledge, ontologies may be used. An ontology is an explicit specification of conceptualization [4]. An ontology is a description of the concepts and relationship among these concepts. Ontologies may describe sets and collections as sets and objects as individuals. Features and characteristics of objects are described as properties. Also, relations are used to describe how objects are related to each other [23].

Ontologies define a common vocabulary between agents, easing the communication, allowing them to use the same semantic mapping [24]. One nice feature of ontologies is that it gives semantics independent of the reader and the context [25]. However, ontologies are not restricted to represent definitions or class hierarchies, they also keep knowledge about the world [26]. A formal ontology guarantees consistency, but not completeness which is realistic, since we cannot always observe and describe every thing that exists in the world.

Ontologies are used by many applications, ranging from recommender systems to e–commerce applications [27]. Given the brief description about ontologies given above, we also prefer using ontologies as our knowledge representation tool. Our agents benefit from ontologies in two main ways. The first is using ontologies as a common vocabulary. They guarantee the same semantics by following the ontologies.

The second is using ontologies to reason about the world with reasoning frameworks such as Jena [28]. By reasoning on ontologies they discover the relations that exist between classes, individuals and properties. Our agents combine the knowledge they gather from the ontology with their initial states and make decisions about entering and proposing contracts.

3.1. System Architecture

The main components of the system are depicted in Figure 3.1. Agents are shown in rectangle nodes. Symbolically only three agents that represent devices and only one UA is illustrated. Obviously, there may be more than three intelligent agents in the system. Also, multiple UA can enter the system in order to serve the user it represents. The ontologies that are used in the system are shown in ellipse nodes. There are two ontologies in the system that represent the relevant information about the world. Line edges describes the two way interaction between agents, whereas dashed edges represent agents' access to the ontologies.

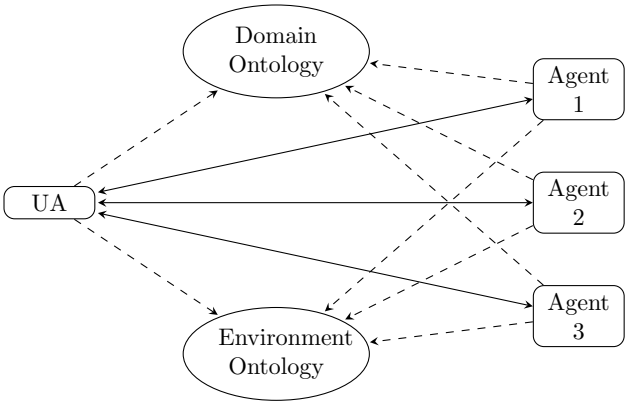

Figure 3.1. Architecture of the system.

UA interacts with all of the agents in the system. UA keeps track of the user's needs and desires and tries to provide the user her preferred set of services. UA is a proactive agent and usually starts the communication. It is also UA's duty to find new communication patterns in order to establish contracts for the services when it fails to establish a contract after interacting with one agent.

Elements of the set of services demanded by the user are often served by various agents, so other agents cooperate with UA. These agents adopt a reactive behavior until the communication is started by UA. When they start to interact with UA in order to establish a contract, they also show more proactive behavior and make requests from UA. These agents are also autonomous, so they make their own decisions for whether or not entering in a contract. The factors that affect their decision are: internal states of the agents, the state of the world and agent characteristics. The internal state of the agent includes its resources to which the agent has access to. Also the ability to provide a service is part of the internal state of the agent. The state of the world includes the service demanded by the agent, resources that are needed to provide this service, external resources to which agent may have access, other agents' willingness to cooperate and all other properties that the agent itself cannot control. The agent characteristics include the agents' willingness to cooperate.

The first ontology is the environment ontology, which describes the environment. The agent, contract and service bundle descriptions as well as additional spatial information about the environment is described in the environment ontology. Although the descriptions for the agent and contract structures are depicted in this ontology, information about individuals are not kept in here. The information not revealed in this ontology is a part of the agent's internal state.

The second ontology is the domain ontology. In this ontology, detailed descriptions of the services and other domain dependent information are provided. These two ontologies are highly related to the application domain. Details of these ontologies for our selected domain will be given in Section 4.

3.2. Contract Life Cycle

In our system, interaction among agents is conducted via messages that create or manipulate contracts between two agents. Contracts are dynamic entities of the system end their states are updated by the agents after receiving or sending certain type of messages. States of contracts used in the system are:

- requested: These contracts are requested from an agent, however the reply for the request has not been received yet.
- rejected: These contracts are the ones that are requested and got a negative respond in return. They do not have any binding effect on either of the parties.
- conditional: These contracts are agreed on and created by both parties. However, their conditions and propositions remain unsatisfied.
- cancelled: These contracts are cancelled by the debtor.
- active: These contracts are agreed on and created by both parties. Moreover; their conditions are satisfied by the creditor.
- released: These contracts are released by the creditor, so the debtor of these contracts are no longer committed to fulfill the propositions of the contracts.
- fulfilled: These contracts are agreed on and created by both parties. Their conditions and propositions are satisfied.

The message types used to carry these contract, their conditions and propositions are listed below:

- request: These messages are used to form a contract, thereby leading the contract to its requested state.
- reject: A reject message changes the contract state from requested to rejected.
- confirm: A confirm message updates the states of the requested contracts to conditional.
- cancel: A cancel message carries a contract that is cancelled by the debtor. The cancel message changes the state of the contract from conditional to cancelled.
- release: A release message carries a contract that is released by the creditor. The release message changes the state of the contract from conditional to released.
- inform: An inform message is used to fulfill the conditions of the conditional contracts (thereby, making the contract active) or the propositions of the active contracts (thereby, making the contract fulfilled).

Figure 3.2 explains the state changes of contracts. A contract is created when it is requested by an agent. If the agent that receives the request rejects the contract its

state is changed to rejected. If the contract is accepted by the other party, its state is changed to conditional. Contracts that are in conditional state. may be cancelled by the debtor agent or may be released by the creditor agent. If the contract is not released or cancelled, and the condition of the contract is provided, it is state is changed to active. When the proposition of the contract is made available by the debtor, its state is changed to fulfilled.

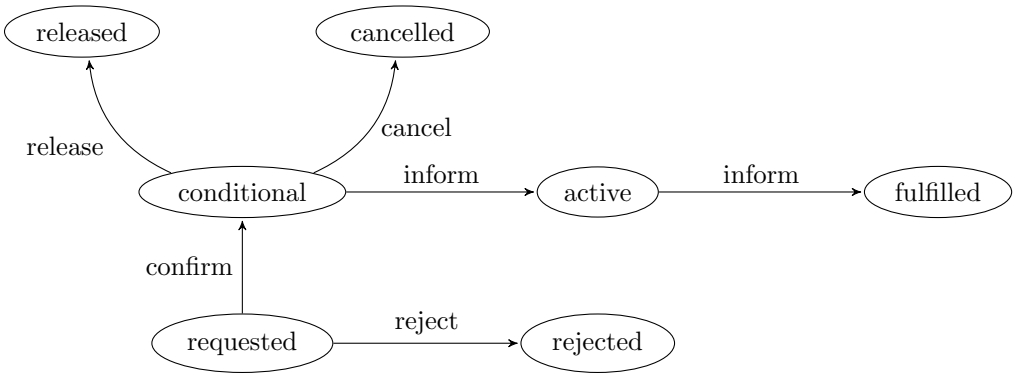

Figure 3.2. Commitment states as nodes and message types as edges.

3.3. Agent Architecture

Figure 3.3 depicts the structure of an agent. Each agent in the system has access to the environment ontology and the domain ontology. So agents share common knowledge about the environment, the context, the user context, services and so on. Ontologies are always extensible, so new classes and object can be added to the ontologies and the agents can update their knowledge when they need to do so.

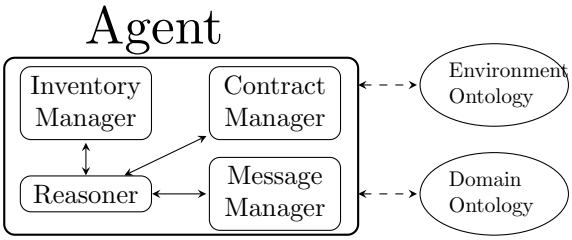

Figure 3.3. Architecture of an agent.

3.3.1. Inventory Manager

Every agent has a local inventory where it keeps the availability information on the service resources. An agent consults its inventory through its inventory manager whenever it receives a service request from UA. The information about the agent's inventory is private and it is not shared with the other agents of the system. The basic operations managed by the inventory manager are:

- Updating a resource: This operation includes both adding a resource to the inventory and decreasing the quantity of the resource when a resource is used.
- Retrieving the quantity of a resource: An agent's reply to a contract request is mainly based on the result of this query.
- Retrieving the quantities of all resources in the inventory: This query is used for monitoring and logging purposes.

3.3.2. Contract Manager

The contract manager of an agent manages the contracts of the agent. It updates the contract states, traces the fulfillment of the propositions and conditions. Each agent handles its contracts itself, so there is not a common contract base of the system as it is not the case in the real life. The operations managed by the contract manager are:

- Adding a contract: This operation is one of the basic operations. It adds a new contract to the agent's contract base.
- Removing a contract: This operation removes a contract from the agent's contract base. A contract may be removed from the system for various reasons based on its establishment time, state and so on.
- Updating a contract: An agent need to update the state of the contract during computation.
- Retrieving contracts that are credited by the agent: This operation is used by agent for monitoring purposes.

- Retrieving contracts that are debted by the agent: This operation is used by agent for monitoring purposes.
- Retrieving contracts that have certain conditions: This operation is used when an agent needs to update a contract based on its conditions.
- Retrieving contracts that are certain propositions: This operation is used when an agent needs to update a contract based on its conditions.
- Retrieving contracts that has certain states: This operation is used by agent for monitoring purposes.
- Retrieving contracts that are part of a certain conversation: This operation is used by agent to retrieve the contracts that are established during a conversation between another agent.

3.3.3. Message Manager

The message manager of an agent manages incoming and outgoing messages of the agent. It labels the messages and directs the message content to the reasoner of the agent when necessary. The operations of the message manager are:

- Sending a message: This is a basic functionality of the message manager.
- Receiving a message: The message manager is responsible of receiving messages, so that other components of the agent continue with their work.
- Passing the message to the reasoner: After receiving a message, the message manager passes the message to the reasoner. The reasoner makes the decisions based on various criteria such as message label, inventory state, ontological reasoning, contract states and so on.

3.3.4. Reasoner

The reasoner of the agent makes the decisions, takes actions and handles messages. Here, we explain a basic understanding of an agent reasoner. Algorithms executed by reasoners of different agents slightly differ from each other. This adds heterogeneity to the system.

Workflow diagram for UA is given in Figure 3.4. When UA tries to establish contracts for a service bundle, it starts with getting addresses of the agents that provides services from the bundle. The agent development framework described in Chapter 4 provides the directory facilities. UA may query the framework for agents that provide certain services. If UA cannot find any agents for one or more services, bundle cannot be served (Failure). If there are agents that serve services of the bundle, UA sends them contracts requests and starts waiting for the replies. Once it receives a confirmation for a contract, it checks whether it gathers confirmation for all contracts it has requested, since our aim is to serve the all services of the bundle together. If there are still some contracts to be confirmed, UA continues to wait for the replies. If all of the contracts are confirmed, UA provides the conditions of the contracts when it wants to get the services from the provider agents. UA's duty ends here as it is the other agents duty to provide the services promised and the exceptions are not in the scope of this work. If UA receives a rejection instead of a confirmation, it searches for other agents that serve the same service immediately. If there are not such agents, UA cannot provide the bundle to the user (Failure). If there are other agents serving the same service, UA repeats the process of requesting contracts. UA may also receive a contract request as a reply for its initial request.

When an agent including UA receives a contract request, it should decide to create it or not. There are three possible reactions that it may take: 1) Rejecting to create the contract, 2) Creating the contract in line with the requester's desire, 3) Requesting another contract that has the same proposition as the contract requested by the requester with a different set of conditions. It is assumed that agents are willing to create contracts unless they lack the necessary amount of resources to serve the service and they do not receive any contract requests beyond their serving capabilities. Thus, when an agent receives a request message carrying a contract, it first extracts the services from the contract. Then for each service requested, it checks whether it is able to serve the service in general. For example, coffee is a service that is served by CMA. When the agents confirms that the service can be served, it checks its current state. The agent may have the service ready to be served when it receives the request. So, it accepts the requests immediately. In case where the agent does

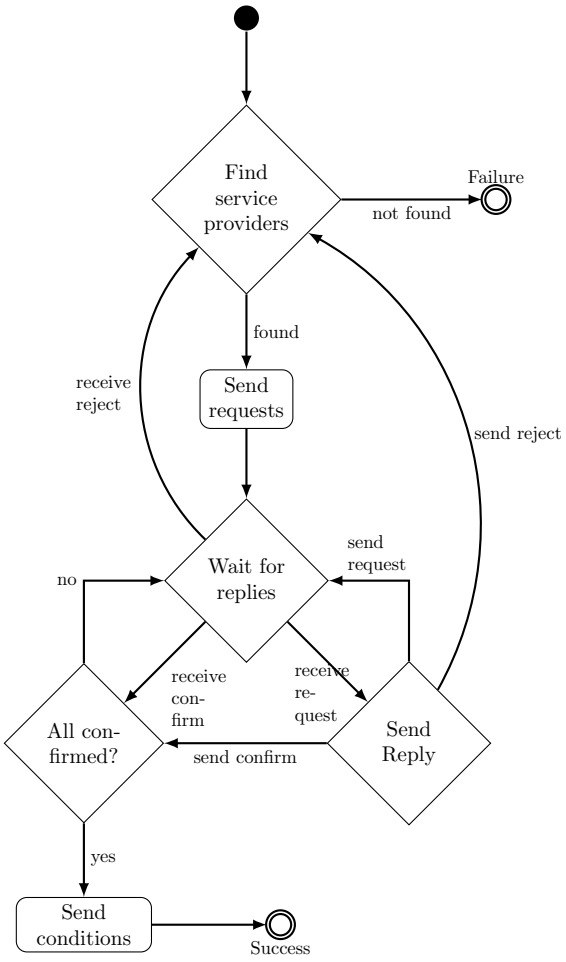

Figure 3.4. Workflow of User Agent.

not have the service ready, the agent should consult the domain ontology to get the resources that are needed to serve the service. Once the agent gets the resources, it should check its inventory to see which resources are readily available to the agent. If the agent has access to all of the resources, it determines that it can serve the service. So, the agent accepts the request. If some resources are missing, the agent may request these resources from the agent, which previously request the service that requires the resources. If the agent is unavailable to serve the service for any other reason , it rejects the request.

Figure 3.5 includes the algorithm which explains the behavior of an agent other than UA when it receives a request message. The message received can start a new conversation between agents, or it might carry on a previous one. So, an agent checks whether the message is part of a previous conversation or not (line 5). If the message is related to a previous contract, it retrieves the contract from its contract base and calculates the similarity between the conditions of the two contracts (line 6). If the similarity is above a threshold set by the agent itself (line 7), it confirms the contract and prepares a confirmation message to be sent to the requester via the message manager of the agent (line 8). If the similarity is below the threshold, a rejection message is prepared instead of the confirmation message (line 10). If the message is not related to any other conversation, the agent checks its inventory for the proposition (line 18). If the proposition is not ready in the inventory (line 19), for this time the agent checks the inventory for the ingredients of the proposition. If there are some missing ingredients (line 21), the agents prepares a request message asking for the missing ingredients in return of the proposition of the contract and returns this message (lines 20-25). Otherwise, the agent prepares a confirm message (lines 26, 27).

When an agents receives a reject message, it should update its contract base accordingly. Figure 3.6 include the algorithm that describes this process. Reasoner contacts with the contract manager to update the contract to the rejected state (line 2). For example, when UA requests the contract CC(UA, CMA, CoffeeRequest, Coffee), it initially generates the contract in requested state in its contract base.

```
   Input: request:Request Message received
   Output: m:Message to send
 1 String id=request.getConversationID();
 2 Contract c=request.getContent();
 3 boolean found=false;
 4 for i ← 1 to contracts.size() do
 5 |   if contracts(i).conversationID==id then
 6 |   |   similarity=getSimilarity(c.proposition, contracts(i).proposition);
 7 |   |   if similarity>threshold then
 8 |   |   |   m.type ← confirm
 9 |   |   else
10 |   |   |   m.type ← reject
11 |   |   end
12 |   |   found = true;
13 |   |   break;
14 |   end
15 end
16 if !found then
17 |   ResourceList rList=c.getProposition();
18 |   ResourceList missing;
19 |   for i ← 1 to gList.size() do
20 |   |   Resource r=rList.elementAt(i);
21 |   |   double invQ=Inventory.getResourceQuantity(r);
22 |   |   if g.RequestedQuantity > invQ then
23 |   |   |   missingResources(g,missing);
24 |   |   |   if missing.size()!=0 then
25 |   |   |   |   m.type ← request;
26 |   |   |   |   c.condition ← missing;
27 |   |   |   |   m.add(c);
28 |   |   |   |   return m
29 |   |   |   end
30 |   |   end
31 |   |   m.type ← confirm;
32 |   |   m.add(c);
33 |   |   return m
34 |   end
35 end
```

Figure 3.5. Request received algorithm for agents.

When the contract is rejected by CMA, the state of the contract is changed to rejected. When the contract state is updated, based on the contract manager's implementation, the contract may be removed from the contract or it may be kept as it is. The reasoner does not bother with the underlying details.

```
   Input: reject: Reject Message Received
   Output: True: Contract updated, False: Update failed.
1  Content c=reject.getContent();
2  return Contract Manager. updateContract(c, rejected);
```

Figure 3.6. Reject received algorithm for agents.

When an agents receives a confirm message, it should update its contract base accordingly. Figure 3.7 include the algorithm that describes this process. The reasoner contacts the contract manager to update the contract state from requested to the conditional state (line 2). For example, when UA requests the contract CC(UA, CMA, CoffeeRequest, Coffee), it initially generates the contract in requested state in its contract base. When the contract is confirmed by CMA, the state of the contract is changed to conditional.

```
   Input: confirm: Confirm Message Received
   Output: True: Contract updated, False: Update failed.
1  Content c=confirm.getContent();
2  return Contract Manager. updateContract(c, conditional);
```

Figure 3.7. Confirm received algorithm for agents.

When an agents receives a cancellation message, it should update its contract base accordingly. Figure 3.8 include the algorithm that describes this process. The reasoner contacts he contract manager to update the contract to the cancelled state (line 2). Consider the case where CMA cancels the previously generated contract CC(UA, CMA, CoffeeRequest, Coffee). When UA receives the cancellation message, it updates the state of the previously generated contract from conditional to canceled. When the contract state is updated, based on the contract manager's implementation, the contract may be removed from the contract or it may be kept as it is. Reasoner does not bother with the underlying details.

```
   Input: cancel: Cancel Message Received
   Output: True: Contract updated, False: Update failed.
1  Content c=cancel.getContent();
2  return Contract Manager. updateContract(c, cancelled);
```

Figure 3.8. Cancel received algorithm for agents.

When an agent receives a release message, it should update its contract base accordingly. Figure 3.9 include the algorithm that describes this process. Reasoner contacts with the contract manager to update the contract to the released state (line 2). Consider the case where UA releases CMA from a previously generated contract, CC(CMA, UA, CoffeeRequest, Coffee). When CMA receives the release message, it updates the state of the previously generated contract from conditional to released. When the contract state is updated, based on the contract manager's implementation, the contract may be removed from the contract or it may be kept as it is. Reasoner does not bother with the underlying details. In addition to receiving a

```
   Input: release: Release Message Received
   Output: True: Contract updated, False: Update failed.
1  Content c=release.getContent();
2  return Contract Manager. updateContract(c, released);
```

Figure 3.9. Release received algorithm for agents.

request message, an agent can also receive an inform message. If that is the case, the agent extracts the messages to get its content and finds relevant contracts through its contract manager. If it finds a contract whose condition matches the content and whose state is conditional, this means that the creditor agent satisfies the condition of the contract. For example, CMA receives an inform message whose content is CoffeeRequest and it has a contract CC(CMA, UA, CoffeeRequest, Coffee) in conditional state. UA satisfies the condition of this contract with the inform message it sends, and CMA should update the state of the contract from conditional to active. Now, CMA is responsible to carry out the rest of the contract by bringing about its proposition, i.e. by providing coffee.

Another case arises when the receiver agent finds a contract whose proposition matches the content and its state is active. Consider the situation where CMA

described above provides Coffee to UA in return of CoffeeRequest. It sends an inform message carrying Coffee to UA. When UA receives this inform message, it checks its contract base and finds the matching contract CC(CMA, UA, CoffeeRequest, Coffee) in active state. The new inform message carries the proposition (coffee) which fulfills the contract. So, UA updates the state of the contract from active to fulfilled.

```
   Input: inform: Inform Message Received
 1 Content c=request.getContent();
 2 for i ← 1 to Contracts.size() do
 3 |   if Contracts(i).condition==c.proposition &
   |   Contracts(i).state=conditional then
 4 |   |   contracts(i).state← active
 5 |   else
 6 |   |   if Contracts(i).proposition==c.proposition &
   |   |   Contracts(i).state=active then
 7 |   |   |   serve proposition;
 8 |   |   |   contracts(i).state ← fulfilled
 9 |   |   end
10 |   end
11 end
```

Figure 3.10. Inform received algorithm for agents.

Agents themselves decide when to cancel and release messages. Policies of canceling and releasing a contract may differ from agent to agent. Some agents may set a time limit and cancel or release the contract when the limit is passed. Some may cancel or release the contracts when they leave the system. Some may do so when their serving capabilities are changed due to various reasons. Since there is a wide number of possibilities, possible algorithms that describe the cancellation and release policies are skipped.

4. IMPLEMENTATION

The proposed architecture is implemented as a multiagent system and specific ontologies are developed for a kitchen domain.

4.1. Agent Development Platform

There are many agent development frameworks that save developers from the burden of implementing the low level details. Before we start implementing our approach, we examined some of this platforms. JADE (Java Agent Development Framework) [29], JACK (Java Agent Component Framework) [30] and Jason (Java Agent Speak Interpreter) [31] platforms are examined to be selected for implementation.

Jason is a Java interpreter for Agent Speak [32]. Agent Speak is an agent programming language for Belief–Desire–Intention (BDI) agents. Jason is open sourced, and allows the implementation of agents using Java language. Agents act based on their plans to reach their goals. There are different types of goals that can be set. One drawback of Jason is that the agents are unable to adapt to the new situations. Initial plans of an agent may not be sufficient to cover all possible situations that the agent may be faced with during computation. Also, the communication between agents do not follow FIPA standards. We have eliminated Jason platform due to its static agent structure.

JACK platform is also aimed for BDI agents. JACK claims to use a model of human reasoning and team behavior in its design. It provides its own Java–based plan language. Jack is not FIPA compliant by itself but it can be with an extension.

JADE also adopts BDI based architecture. It provides a distributed and steady system which is appropriate for our work. The programming language is Java, which can be seen as both an advantage and a disadvantage. Some may argue that this

lacks the reasoning power of logic based or agent based programming languages; but we take the advantage of pure Java language, its free nature that does not force us to use any engineering technique, its platform independence and also skipping to learn a new language and being aware of the fact that we can implement a system as powerful as in other languages, we chose JADE.

JADE provides a distributed system where containers can run on different servers. Agents run in these containers. Agents not only communicate with agents that are in the same container, but they can also communicate with agents that run in other containers.

JADE provides a Directory Facilitator (DF) which provides yellow pages services. Agents may query the DF to discover other agents according to their names or services. Agents register themselves and their services when they enter the system using Agent Management Service. JADE also handles the communication between agents. Since it is FIPA compliant, it provides interoperability among different applications.

The attributes of JADE messages that are used in our application are:

- Performative: The type of the message. The message types used for this application is described in Section 3.2.
- Sender: The agent that sends the message
- Receiver: The agent that receives the message
- Conversation ID: The identification that used to mark messages in the same conversation.

Other attributes of JADE messages such as language, protocol, replyWith and so on are not used for our system.

JADE has concept of behaviors which are adapted by agent and executed according to their types. Cyclic behaviors are executed over and over whereas one shot

behaviors are executed only once. Agents may activate some behaviors during run time, so they are not static structures. In our implementation we put our algorithms in separate behaviors and activate the relevant behavior when it is needed.

4.2. Ontology Development

While developing our ontologies, we chose OWL [33] because OWL is an expressive formalism language. It is has and RDF and XML syntax which allows ontologies to be transferred to other forms easily when desired. OWL is expressly designed as an ontology language, and it has many predefined classes and properties useful for expressing ontological information. It is also a W3C standard for web ontologies.

We used Protégé platform to develop our ontologies. Protégé is an open source ontology editor by Stanford University. It is based on Java and can be extensible by plug-ins. The version used is Protégé 4.1, which has full support for OWL 2.

We apply our approach on an AmI kitchen domain. An AmI kitchen consists of various autonomous agents such as Coffee Machine Agent (CMA), Tea Machine Agent (TMA), Fridge Agent (FA) and Mixer Agent (MA), which represent devices in a regular kitchen. Each of these agents provide different services. Agents use some ingredients related to their services as resources. For example, CMA, which serves coffee, has coffee beans and water in its inventory. It may also have some coffee ready in its inventory. Similarly, TMA which serves Tea is expected to have tea leaves and water. On the other hand, FA has some cake to serve. UA of the system tries to serve the user a service bundle which is a menu consisting of several beverages and dishes for this domain. Each element of a menu is usually served by a different agent of the kitchen.

The user of the system is satisfied when she gets the exact menu she prefers. Establishing contracts is a necessity in such a system for user satisfaction since the static contracts will not work for the reasons described in Section 2. Agents of the system may get broken, broken ones may be fixed or replaced, or new agents may

enter to the system so the assuring power of the predefined contracts established between agents is limited. The availability of the resources is limited, so the agents do not always have access to the resources they need.

4.2.1. Environment Ontology

The environment ontology of this system describes the agent structure, contract structure and spatial information about the kitchen such as the temperature and humidity level. Our kitchen has the following attributes:

- hasHumidity: This property shows the humidity level in the kitchen.
- hasTemperature: This property shows the temperature level in the kitchen.
- hasAgent: This property links the agents in the system with the kitchen.
- hasUser: This property links the user of the system with the kitchen.

Other than kitchen class, we have an agent class. The properties of the agent class are:

- hasName: This property indicates the identifier of the agent.
- hasInventoryManager: This property indicates that each agent has an inventory manager. Cardinality of this property is exactly one.
- hasContractManager: This property indicates that each agent has a contract manager. Cardinality of this property is exactly one.
- hasMessageManager: This property indicates that each agent has a message manager. Cardinality of this property is exactly one.
- hasReasoner: This property indicates that each agent has a reasoner. Cardinality of this property is exactly one.
- serves: This property links the agent with the services it is able to serve. There may be zero to many services that are linked to an agent.

Contracts used int he system are also described in the ontology. Properties of the contracts are:

- hasDebtor: Each contract has an agent that is committed to bring out the proposition of the contract.
- hasCreditor: Each contract has a creditor agent.
- hasCondition: Each contract has a condition. If the debtor agent promises to satisfy the condition in any condition, condition of the contract is True.
- hasProposition: Each contract has a proposition that is promised to be satisfied by the debtor agent.
- hasState: Each contract has a state. The value of this proposition is limited to the contract states described in Section 3.2.
- hasConversationID: Each contract is a part of a conversation between two agents. We use a conversation ID in order to distinguish between these conversations.

The user of the system is also described in the environment ontology. In our application, we have not not dived into very much detail about modeling the user. The approach we developed and the domain we work on can be used for various applications. In a medical oriented application, the user class should include the allergies of the user, her medical condition, the dishes that should be avoided by the user, the dishes that should be consumed by the user and so on. For a more gastronomy oriented application, the personal preferences of the user should be in the ontology. Here we propose a model that can be used as a base for various applications.

4.2.2. Domain Ontology

The domain ontology of this environment is a food ontology, in which various types of food and beverages together with their ingredients are described. Agents use the recipes provided in the ontology for their services. In this ontology. the ingredients and types of some most popular items such as coffee and tea, are carefully classified and some similarity factor is placed between pairs that are substitutable. The similarity factor shows how well these items can substitute each other. Higher the similarity factor is, stronger the similarity relationship between the items that are compared to. These similarity factors are used to serve the demanded dish with a

slightly different recipe when the original ingredients are not available in the inventory of the agent and UA cannot establish a contract that promises the missing ingredient. In such cases, the agent may try to prepare the dish using the substitute of the missing ingredient.

Let's consider three types of Flour that are classified under Wheat Flour class. These types are All Purpose Flour, Cake Flour, and Bread Flour. All Purpose and Cake Flour are 0.7 similar, whereas Cake Flour and Bread Flour are 0.8 similar. When a service which requires one of these types of flour is requested, and the exact resource is not available, the resource that are similar may be substituted by one of the other types, leading to the same service served with tolerably different resources.

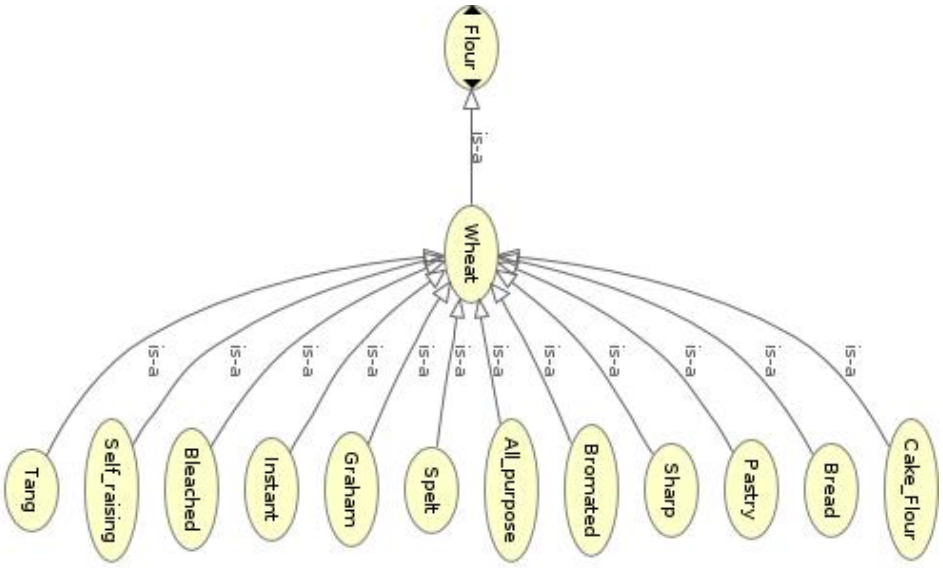

Figure 4.1. A caption from example domain ontology, representing flour class.

In case the substitute being missing too, it may tray to settle a contract for the substitute. For example, Tea main entity has three children which are Green Tea, Black Tea and White Tea. Agents need to have corresponding tea leaves type, i.e. green tea leaves, black tea leaves or white tea leaves, and some water to serve any of them. Green Tea and Black Tea shares a similarity factor of 0.5. On the other hand, Black Tea and White Tea have a similarity factor of 0.8 which means that Black Tea is more similar to White Tea than it is to Green Tea. Note that not all items need to have similarity factors with every other item in the ontology. For this particular case, White and Green Tea do not have a similarity factor. The food ontology is beneficial when there is no way to successfully serve the exact item the user prefers.

For this case, UA tries to serve the item that is most similar to the preferred item. For example, when the user agent fails to get Black Tea from agents by all means, it may switch to White Tea which is the most similar item to Black Tea in the ontology.

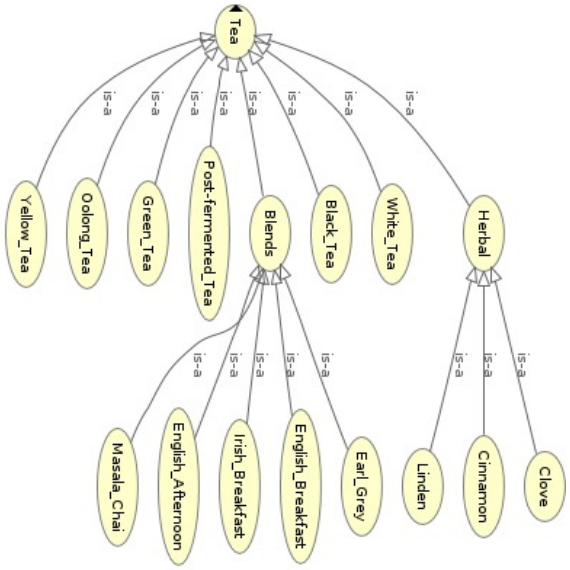

Figure 4.2. A caption from example domain ontology, representing tea class.

The detail level of a domain ontology changes from system to system. Agents of another kitchen may use a domain ontology just for the ingredients without the similarity relationship. Another one may also include the types of silverware that should be used with a specific dish.

5. CASE STUDY

We now study different cooperation scenarios to understand how well our system accommodates flexible interactions. For each scenario, we first describe the situation and the system state, then provide the execution of our system. The flow of communication is illustrated for each execution.

5.1. Scenario 1: UA Provides a Missing Resource

The initial state of the system for the first scenario: There is one user that uses the system. UA represents this user. There are two agents, namely, CMA and FA. CMA serves coffee and FA serves cake. In the domain ontology, it is stated that coffee beans and water is needed to serve coffee. FA has a cake ready to be served in its inventory; however CMA lacks coffee beans to serve coffee. UA has access to coffee beans. The inventories of agents are not public knowledge.

The user tells UA that she wants a menu consisting of coffee and cake, which should be served together. First of all, UA should decompose the request of the user. It should recognize that two distinct services are requested. UA should discover agents that serve the requested services. In this case these agents are CM and FA. UA should persuade these agents to serve coffee and cake, respectively.

5.2. Execution of Scenario 1

Figure 5.1 depicts the execution of the scenario described in Section 5.1. For simplicity, agent names are omitted from the contracts. In order to realize the scenario, UA sends request messages to start conversation (M 1 and M 2). At this point, UA has the two contracts in requested state in its contract base. After FA sends a confirmation back (M 3), the state of the contract carried by M 3 is changed to conditional. On the other hand CMA is in need of some coffee beans, so it sends a request message back (M 4). When UA receives this message, it drops the previously

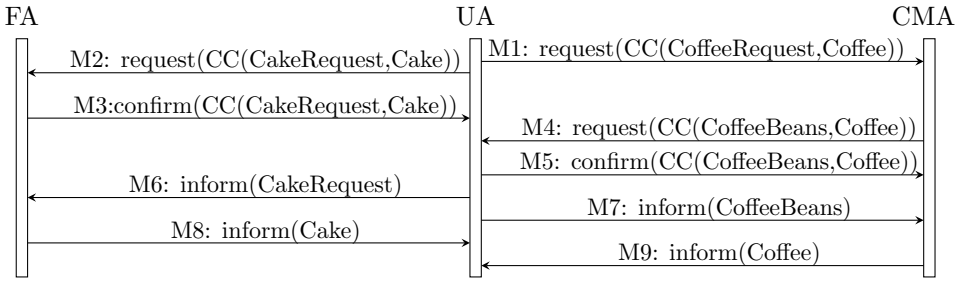

Figure 5.1. Sequence Diagram for Scenario 1.

requested contract. After UA accepts this offer (M 5) the contract carried by M 5 is updated to conditional state in UA's contract base. By accepting CMA's request, UA establishes all contracts necessary to serve the menu. It sends an inform message to realize the condition of the contract with FA (M 6) changing the state of the contract from conditional to active. It also sends an inform message to deliver the condition of the contracts with CMA (M 7). FA and CMA send the propositions of the corresponding contracts (M 8, 9) and the states of the contracts are updated to fulfilled.

5.3. Scenario 2: A Substitute is Proposed instead of the Original Resource

The initial state of the system for the second scenario: There is one user that uses the system. UA represents this user. There are two agents, namely, CMA and MA. CMA serves coffee and MA serves cake. In the domain ontology, it is stated that coffee beans and water is needed to serve coffee. Also cake flour is one of many ingredients that are needed to serve cake. CMA has all the ingredients to serve coffee. MA lacks cake flour to serve cake. UA has access to bread flour. The inventories of agents are not public knowledge.

For the second scenario, the user declares that she wants coffee and cake menu. Again, UA should decompose this request to two different services and discover agents that are capable of serving them, which are CMA and MA. UA should persuade them to serve the requested services. Agents should carefully decide whether they can serve the requested service or not.

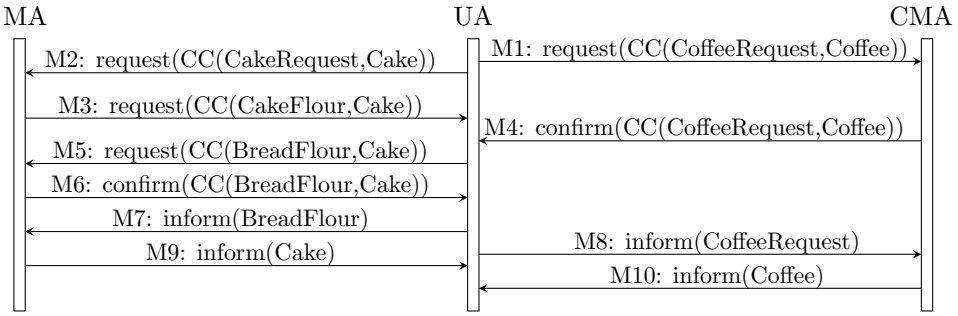

Figure 5.2. Sequence Diagram for Scenario 2.

5.4. Execution of Scenario 2

For the scenario described in Section 5.3, the flow of communication is depicted in Figure 5.2. UA sends relevant request messages to start conversation (M 1 and M 2), so the contracts are created in requested state in its contract base. Mixer Agent immediately makes a request for cake flour, since it does not have the necessary amount of flour to bake the cake (M 3). After receiving this message, UA drops the previously generated contract. Unfortunately, UA cannot provide cake flour, but it consults the domain ontology for the most similar item and it finds out that it is the bread flour and luckily, it can provide bread flour, so it makes a contract request back with bread flour as condition and cake as proposition (M 5). Again, the contract is generated in requested state in its contract base. The substitution satisfies MA and it accepts to take part in the contract (M 6), and the state of the contract is updated to conditional. So, UA establishes all contracts that it needs to do, since CMA has already accepted the request with M 4. UA sends inform messages to both agents, satisfying the conditions of the contracts (M 7, 8) and updating their states to active. After receiving the conditions, agents serve the propositions of their contracts (M 9, 10) and makes the contracts fulfilled.

5.5. Scenario 3: The Substitute is not Accepted

The initial state of the system for the third scenario: There is one user that uses the system. UA represents this user. There are three agents, namely, CMA, MA and FA. CMA serves coffee, MA and FA serve cake. In the domain ontology, it is stated that coffee beans and water is needed to serve coffee. Also cake flour is one

of many ingredients that are needed to serve cake. CMA has all the ingredients to serve coffee. MA lacks cake flour to serve cake. FA has a cake ready to be served in its inventory. UA has access to bread flour. MA requires substitutions for the ingredients to be very similar to the original ones.

For the third scenario, the user declares that she wants coffee and cake menu. Again, UA should decompose this request to two different services and discover agents that are capable of serving them. UA should construct a cooperation mechanism between the agents of the system to serve the bundle. None of the agents are forced to cooperate.

5.6. Execution of Scenario 3

Communication flow between agents for the scenario described in Section 5.5 is represented in Figure 5.3. The scenario starts with UA's sending contracts requests to service providers MA and CMA (M 1, 2). CMA sends a confirmation (M 3) whereas MA requests another contract, demanding cake flour to provide cake (M 4). Unable to provide cake flour, UA requests yet another contract, offering bread flour to get some cake from MA (M 5). MA does not find bread flour similar enough to cake flour, so it rejects the contract offered by UA (M 6). UA searches for another agent that can provide cake service, so it discovers FA. It makes a request (M 7) and receives confirmation in return (M 8). With this confirmation, UA gets confirmation for all contracts to get services for the cake and coffee bundle, so it fulfills the conditions of the contracts (M 9, 10). FA provides the service it is committed to serve (M 11), however CMA gets broken and cannot provide the service. After a certain time, UA gives up hope on CMA and starts looking for a new agent to provide the same service.

5.7. Scenario 4: A New Agent Enters the System

The initial state of the system for the fourth scenario: There is one user that uses the system. UA represents this user. There are three agents, namely, CMA and

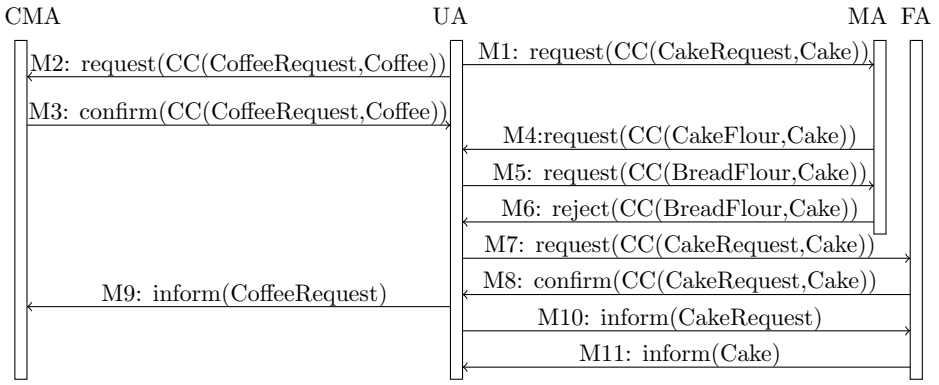

Figure 5.3. Sequence Diagram for Scenario 3.

FA. CMA serves coffee, MA serves cake. In the domain ontology, it is stated that coffee beans and water is needed to serve coffee. CMA has all the ingredients to serve coffee. FA has a cake ready to be served in its inventory. CMA stops serving in run time when it needs to be cleaned. Later a new agent, CMA 2 enters the system. CMA 2 serves coffee.

For the fourth scenario, the user declares that she wants coffee and cake menu. Again, UA should decompose this request to two different services and discover agents that are capable of serving them. UA should construct a cooperation mechanism between the agents of the system to serve the bundle. None of the agents are forced to cooperate. Agents may get out of the cooperative agreement when their internal states change.

5.8. Execution of Scenario 4

Communication flow between agents for the scenario described in Section 5.7 is represented in Figure 5.4. UA sends a request message including a contract which says that CMA serves coffee when a coffee request is made by UA (M 1). UA also sends a similar request message to FA, which carries a contract stating that FA serves cake when a cake request is made by UA (M 2). Both CMA and FA confirms the requested contracts (M 3,4). CMA sends a cancel message which carries the conditional contract generated after M 4 (M 5). Contracts that are in conditional state are the contracts that are confirmed but their conditions are not satisfied. After UA receives the cancel message from CMA, it sends a request message to CMA2 which can serve coffee (M

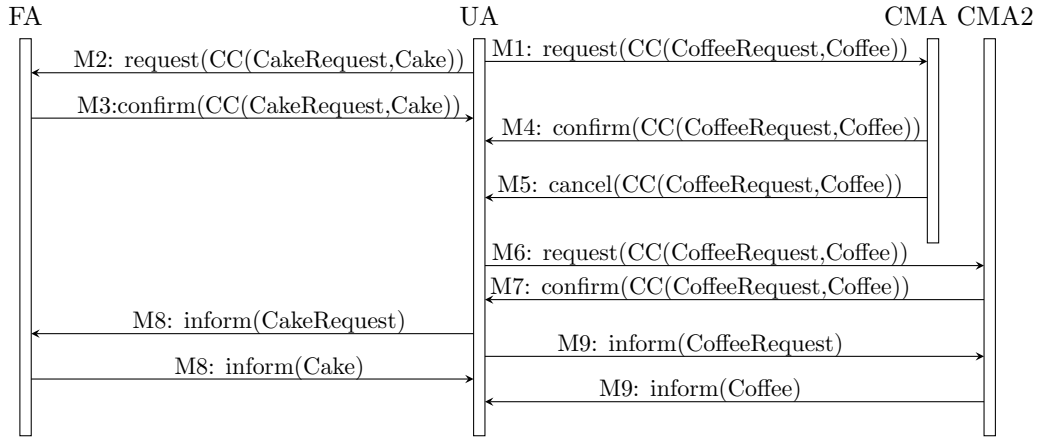

Figure 5.4. Sequence Diagram for Scenario 4.

6). CMA2 sends a confirm message, changing the state of the contract from requested to conditional (M 7). When UA decides to get the services from the agents, it sends inform messages to the agents by satisfying the conditions of the generated contracts (M 8,9). These messages changes the states of the contracts from conditional to active. Once the agents receive the inform messages, they provide the services and the contracts become fulfilled.

5.9. Scenario 5: The User Changes Her Mind

The initial state of the system for the fifth scenario: There is one user that uses the system. UA represents this user. There are three agents, namely, CMA and FA. CMA serves coffee, FA serves cake. CMA has all the ingredients to serve coffee. FA has a cake ready to be served in its inventory.

For the fifth scenario, the user declares that she wants coffee and cake menu. Again, UA should decompose this request to two different services and discover agents that are capable of serving them. UA should construct a cooperation mechanism between the agents of the system to serve the bundle. The user may change her mind and inform UA. UA should stop the process of serving the services not to waste the resources.

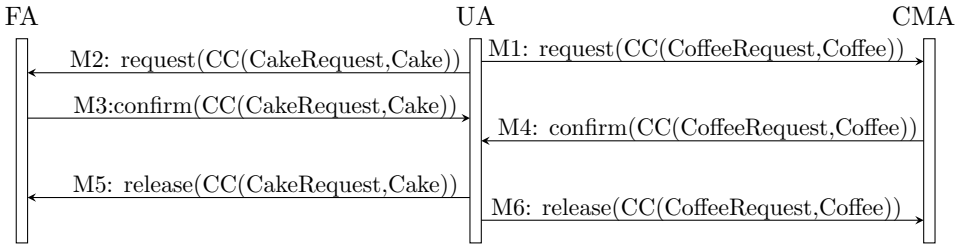

Figure 5.5. Sequence Diagram for Scenario 5.

5.10. Execution of Scenario 5

Our fifth scenario starts with UA's sending request messages to CMA and FA. CMA serves coffee and it receives a request message carrying a contract indicating that CMA serves coffee when coffee is requested (M 1). FA serves cake and it receives a request message from UA which states that fridge serves cake when requested (M 2). Both agents send confirm messages, which changes the state of the contracts that they carry from requested to active (M 3,4). When UA decides to release the agents, it sends release messages that carry the corresponding contracts to each agent (M 5,6). These release messages change the state of the contracts from active to released.

5.11. Scenario 6: FA Adds a New Service

The initial state of the system for the sixth scenario: There is one user that uses the system. UA represents this user. There are two agents, namely, CMA and FA. CMA serves coffee, FA serves donut and orange juice. CMA has all the ingredients to serve coffee. FA has orange juice and donuts ready to be served in its inventory.

For the sixth scenario, the user declares that she wants coffee and cake menu, both items together and nothing else. Again, UA should decompose this request to two different services and discover agents that are capable of serving them. Obviously, it will not be able to find an agent serving cake. Later, the user buys and puts a cake into FA. Since agents can announce their added services, UA can discover FA for the cake service. Then it can construct a cooperation structure among agents to serve the menu.

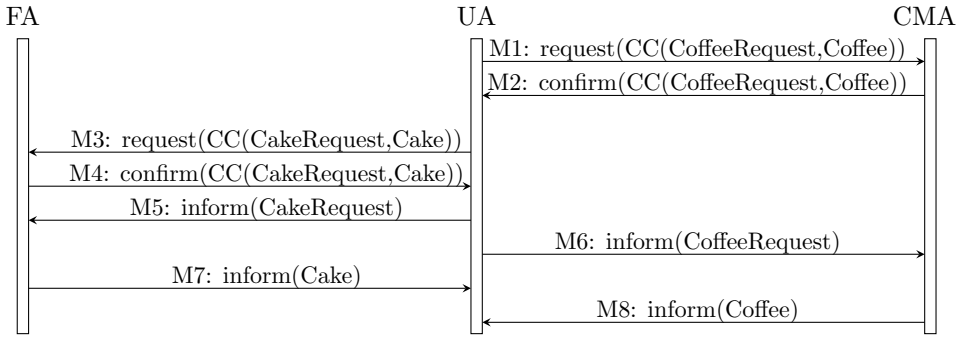

Figure 5.6. Sequence Diagram for Scenario 6.

5.12. Execution of Scenario 6

Our sixth scenario starts with UA's sending a request message to CMA (M 1) and CMA confirms the request (M 2). It keeps checking periodically to find an agent that serves cake. When FA gets the cake, UA discovers that FA can now serve cake. It sends a request message to FA (M 3). FA confirms the request (M 4). As UA collects all contracts from the provider agents, it fulfills the conditions (M 5, 6) and gets served (M 7, 8).

6. RELATED WORK AND DISCUSSION

In this chapter, we investigate some AmI projects that include agents. Then, we summarize our contribution to the literature. We compare our work with the rest and present our future work ideas.

6.1. Related Work

iDorm [19] is designed to provide a smart dormitory room for college students. The application combines agent architecture and network topologies. The architecture of the system is centralized, and the main component is an agent that is associated with the building. The system is pervasive and ubiquitous, yet the user can also intervene the system when she is not satisfied with the results. The embedded agents that are utilized to control the temperature and light of the environment learn from such interventions in order to increase the user satisfaction. The system is also context aware for such controls. As the agents learn from user, the system becomes more and more invisible to the user. iDorm application does not use any of the knowledge representation methods. Also the communication between agents is not FIPA compliant. The system is not flexible in terms of different communication structures between agents and the agents' leaving and entering the system. Also adding or removing services from agents is not supported.

EasyMeeting [20] is developed to manage smart meeting rooms from controlling the lights and speakers to greeting the participants and displaying speaker's profile on the registered browsers. It uses Context Broker Architecture (Cobra) [34] to collect the context information. Context information includes both the environmental information and user context. Bluetooth enabled devices carried by users are used to identify their owners. Information servers such as online calendars are used to gather user context. Several Semantic Web technologies are used to represent and reason on the contextual information. Ontologies are represented in OWL language. SOAP, RDF and OWL links agents and devices to the Context Broker Architecture. Some

of the user information is kept using FOAF. Intelligent agents of the system use Jena framework to reason on the data. There are logical inference rules so the system is not flexible for situations that are not anticipated. Privacy is one of the main concerns of the system. Privacy policy of the system is set using the Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) [35]. Some other components of SOUPA is imported to keep spatial and time related information. Communication between agents and context broker is FIPA compliant and implemented on JADE [29].

MyCampus [36] does not position itself as an AmI environment; however it has some common characteristics with AmI applications. It is a context aware system that aims to controls access to the user's personal preferences and contextual attributes. Users' contextual information is gathered through web services and the mobile devices. Users' preferences and also concerns about security and privacy are taken into account and issues related to these topics are handled. There are agents that are associated with users. Also, some agents manage the public or semi public services of the campus. OWL [33] is used to represent the knowledge, DAML-S [37] is used to reason on this knowledge. Agents of the system do learn their users' preferences and act accordingly. There are different layers of application such as Platform Manager, which offers directory services and User Interaction Manager which manages authentication services. Mobile agents are used to satisfy the user with additional tasks of her choice.

ASK-IT [18] aims to assist elderly and impaired people. It makes an extension to Im@gine IT [38] and Crumpet [39], combining FIPA PTA [40] architecture with OSGI [41]. It is a multiagent system that uses argumentation based interaction in an ambient intelligence context. Elderly and disabled assistant agents, that are specialized on different types of impairments, are used to make distributed decisions. There is an agents which establish coalition between elderly and disabled assistant agents. There are also various types of agents for context retrieving, user monitoring, service provision and so on. The contextual information consists of the user's medical condition and weather conditions. It is mentioned that Jena Framework is used to reason on the knowledge, however the type of the knowledge and how it is represented

is not mentioned clearly. The system is rule based, so it is not flexible for unexpected situations. JADE Framework [29] is used to develop agents.

DALICA [17] project is a multiagent system that monitors user behavior and coordinates user activity in a local historical sight. It helps tourist to visit the historical area providing them guidance and information about the points of interests. It also monitors user behavior in case of bad conduct against point of interests. Another aim is to monitor the transportation of the historical artifacts. It also sets up a user profile and tries to match the users that have common interests. It also aims to plan activities for the user with common activities. There is an ontology to describe the points of interests. DALI [42] is used for the implementation. System works according to defined scenarios and is not flexible. The system is not scalable. The system is not concerned about the environmental or the user context. The communication between agents are not FIPA compliant.

The SpatialAgents platform [43] takes location information as context information. Users' location information is gained through their mobile devices with the use of the RFID tags and there are several Location Information Servers (LIS) that utilize this information. When a new user appears in the range of a LIS, the server sends a mobile agent to execute on the mobile device of the user. The user can then make use of the mobile agent. The platform may host agents that serve various services, it does not have a selected domain. When the user goes out of the range, the mobile agent returns back to the server. There is no cooperation between the mobile agents, also any context information other that the location is not mentioned to be used by the agents. Context information other than the user's position information is not gathered and used. The system is not distributed. It is not flexible for the unexpected cases. The system does not represent knowledge formally, so the ontologies are not used for the system. The system is not distributed. FIPA standards are not used for communication.

The LAICA project [16] proposes an AmI framework that covers a whole city. There are various type of agents in the system, although their roles are not described

in detail. The agents can be as simple as sensors. There are more complex agents that process data gained from such sensor agents. The middleware relies on collaboration rather than cooperation between agents. The middleware allows peer-to-peer connections between agents when it is overwhelmed with data. Some tasks of the middleware can be delegated to the agents that have enough computational power. It is highlighted that for a high scale AmI, not all parts of the system should be intelligent. The key point is that the environment should be perceived as intelligent by the user. Knowledge representation is not mentioned for the project. So, ontologies are not used. Agents do not learn from the user or any kind of feedback. Although sensors are used to gather data, context awareness is not handled. The system is scalable, however not flexible. The communication standards do not satisfy FIPA standards. The system is partly distributed.

AmbieAgents [44] infrastructure is developed as a part of the AmbieSense project [45] which has started as a European Union fifth framework project (U-IST 2001-34244 AmbieSense) and jumped into the commercial market. The infrastructure gathers five different types of context information: social, task, personal, environmental and spatio-temporal. There are context agents which controls the context information to and from its user, recommender agents which determines the user's situation and the content agents which enables the content delivery to the user. Agents are built on the JADE framework [29]. An ontology is developed to describe the main concepts of an airport. The system makes recommendations to the user according to its profile information and the current context. The system is based on the contextual information and learning is not involved. The system is scalable and partly distributed.

The CAMPUS framework [46] tries to built a system that is decentralized. There are three different layers: context-provisioning, communication and coordination, and ambient services. The context-provisioning layer is implemented using MoCA (Mobile Collaboration Architecture) [47]. Multiagent infrastructure is responsible for the transportation of the FIPA compliant ACL messages. There are four types of contextual information: system context, physical context, user context and

time context. There is a campus ontology that describes the domain and the services. The system is distributed and scalable. The framework uses OWL to represent knowledge. The framework does not specify a domain but it rather offers a platform to develop multiagent AmI applications. Many problems such as messaging, context broking and so on are handled.

SodaPop [48] is an infrastructure for AmI that proposes a two stage approach for multiagent AmI systems. Rather than providing a high level software infrastructure, SodaPop aims to provide core facility for self organization appliance ensembles. The infrastructure proposes solutions for communication between devices, distributed system, implementation and ad-hoc discovery of devices. Knowledge representation is not a concern of the infrastructure. It is flexible and scalable. Learning and context awareness is not mentioned. The communication between devices are not FIPA compliant.

In [15], adductive logic based agents are proposed for an AmI elderly care and assist system. There is a medicine manager and a diary/appointment manager. The medicine manager keep track of the medicine take by the user, possible food-medicine conflicts and advises the user accordingly. Similarly, the diary/appointment manager records the appointments with the doctors, anticipates possible conflicts and suggests new dates when necessary. The work focuses on the reasoning on the data gathered. The data consist of the information from the sensors and other sources, as well as knowledge represented in modules, previously defined rules and restrictions, interaction policies and the past data from sensors. Agent autonomy is restricted. Agents collaborates with each other based on previously set collaboration structures.

In [49] autonomous agents form a multiagent system to realize an AmI kitchen. In this research, agents have goals and proactively generate contracts dynamically in run time to satisfy their goals. The system does not represent knowledge, and lack of learning mechanisms. However; it is flexible and scalable. FIPA standards are followed for agent communication. To the best of our knowledge, this is the only system that brings the goal concept into AmI as for our knowledge.

6.2. Contributions

Our thesis proposes a contract based cooperation for ambient intelligence. We describe how such a system should be: Two ontologies are utilized keep the environmental and domain knowledge. The user is represented by an agent of the system. We chose the smart kitchen domain to implement our approach and modeled the devices as intelligent agents. We provide realistic scenarios.

One of our main contribution is that we do not force rigid collaboration structures. In the compile time, we do not load any obligations to any of our agents. Such static obligations may lead to failure since they cannot reflect the dynamism of the system both in terms of environmental changes and state changes of the agents. We design a system that is flexible so it is more prone to the unexpected situations such as agents leaving the system, scarce resources and so on. Since static collaboration structures fail for most scenarios as described in Chapter 2, our agents form short time cooperation themselves in run time. In order to arrange such cooperation, they make use of contracts. Using contracts is one way to regulate multiagent systems. The innovative approach is that the contracts are generated during run time, taking both agents' approval. We need to take both parties since the state of the system and the internal state of the agent affect the agent's ability to serve as described in Chapter 2. In order to handle the dynamism in the system, agents should decide whether or not take part in the contracts.

Such decision making leads us to the second main contribution of our work. The decision on entering a contract is made by reasoning on the knowledge that is relevant and accessible by the agent. Our agents use the domain ontology and their internal state to make such decisions. The knowledge related to services are kept in the domain ontology. By reasoning on this ontology and adding their internal state information, agents decide whether they can provide the service or not. Among the systems investigated in Section 6.1, all of the systems that use ontologies for reasoning, use the results of the reasoning on how the services should provided. It is assumed for sure that the agents are able to provide the service regardless of their

states. However, we do not find this approach realistic and extend it to cover more realistic scenarios. Our agents first make the decision on whether they can serve or not, and then, if necessary they reason on how they should serve. This adds our system flexibility to handle situations described in Section 2.2.

6.3. Discussion

Table 6.1 summarizes some key features that AmI systems should have. The first column is reserved for knowledge representation. Among the applications investigated, iDorm, SpatialAgents, SodaPop, LAICA ASK-IT and [15] does not represent knowledge in any form. MyCampus and AmbieAgents platforms use case–based reasoning (CBR) modules are used to represent knowledge, whereas Dalica project makes use of tuples for the same purpose. EasyMeeting, CAMPUS and this thesis benefits from ontologies to represent knowledge.

The second column is devoted to use of ontologies. Projects that use ontologies are EasyMeeting, MyCampus, AmbieAgents, CAMPUS, Dalica and this thesis. EasyMeeting uses SOUPA for setting privacy and security preferences of the user. MyCampus, and AmbieAgents keep contextual information in the ontologies. On the other hand, CAMPUS models its system in its ontology. DALICA both models its system and keeps contextual information in ontologies. This thesis also uses ontologies to model its system and keep the contextual information.

The third column represents the context awareness of the systems. As described in Chapter 1, context awareness briefly means being aware of and responding to the changes in the environment and the user. The term has a broad meaning, since these changes range from perceiving the existence of the user to accessing her daily schedule. So, the level of context awareness may vary widely among the projects. For example, SpatialAgents platform is only aware of the location information of the user. MyCampus considers the schedule of the user and weather forecast as context information. Dalica takes the locations of the historical artifacts as contextual information. The system developed in this theses considers the environmental data

and the user's needs and desires as contextual information. Therefore, we can say that it is context aware.

The fourth column displays the systems which learn and improve themselves from the user feedback. iDorm, SpatialAgents, MyCampus, ASK-IT, and CAMPUS takes user feedback as input and improve the system. iDorm learns from its user in order to take some actions such as closing the curtains when the light level increases.

The fifth column is related to the scalability of the systems. We are not provided with enough information about the scalability of iDorm, SpatialAgents and Easy Meeting systems. ASK-IT, Dalica and [15] are not scalable systems. For example in [15] there is only one agent that use argumentation to serve its user. The scalability of JADE is discussed in [50]. Although there are some limitations such as the number of agents that can be created in a container cannot exceed the number of threads that can be created within a Java program, it is stated that these limitations can be overcome using certain methods. Since we used JADE as our agent development framework, we consider the system developed in this thesis as scalable.

The sixth column reflects the flexibility attributes of the systems. SpatialAgents, EasyMeeting, SodaPop, MyCampus, and CAMPUS systems as well as this thesis are flexible systems which can increase the number of services provided to the user. New agents with serving various may enter in to the systems and enhance the systems' overall capability of serving the user.

The seventh column shows the systems are organized. This thesis, CAMPUS and SodaPop are distributed systems. On the other hand, LAICA, AmbieAgents and DALICA are partially distributed systems, they have some centralized modules. Other systems have a centralized structure.

The last column indicates the FIPA compliance of the systems. EasyMeeting, AmbieAgents, ASK-IT, CAMPUS, and this thesis follow FIPA standards for communications whereas other systems use their own structure for communication between

agents.

Table 6.1. Comparison of the related work.

Project Name	knowledge representation	ontologies	context awareness	learning	scalability	flexibility	distributed	FIPA-compliant
iDorm	-	-	+	+	?	-	-	-
SpatialAgents	-	-	+	-	?	+	-	-
EasyMeeting	Ont.	+	+	-	?	+	-	+
SodaPop	-	-	+	-	+	+	+	-
LAICA	-	-	+	-	+	-	partial	-
MyCampus	CBR	+	+	-	+	+	-	-
AmbieAgents	CBR	+	+	-	+	-	partial	+
ASK-IT	-	+	+	-	-	-	-	+
CAMPUS	Ont.	+	+	-	+	+	+	+
Dalica	Tuples	+	+	-	-	-	partial	-
[15]	-	-	+	-	-	-	-	-
[49]	-	-	-	-	+	+	+	+
This Thesis	Ont.	+	+	-	+	+	+	+

6.4. Future Work

Future work may include adding a recommendation component to the reasoner of UA, so that UA can make suggestions or improvements to the service bundles requested by the user. Such recommendation system should take into account the history and preferences of the user, as well as the system and user context. For example, if we decide to add a recommender system for medical reasons, UA should consider the current medical condition of the user, such as test results, current blood sugar level, current heart rate and so on, the constraints on menus such as not taking alcohol within two hours of taking a certain medicine, the preferences of the user and the contextual information such as the weather. For instance, a piece of cake may be served with cold lemonade in a hot summer day, whereas a cup of tea may be a more suitable option for a cold winter day. Past decisions and the satisfaction of the user may be used to train the recommender system.

Allowing inter–communications between different AmI system is also a research topic that should not be neglected. JADE allows for such communications; however, before letting two systems communicate, some issues such as security and privacy should be handled. There may be added some new agents to the system dealing with such issues. The preferences of the user should be taken into account and the other AmI system should be informed about the preferred policies. Ontologies can be used for such knowledge sharing and a security–privacy ontology may be developed, or an existing one such as SOUPA [35] may be adopted and extended for this purpose. Inter–communication between two systems may result in interesting scenarios to handle such as planning a party together or ensuring that the one user is served according to her preference when she visits the other AmI. These are interesting topics that we defer to future work.

REFERENCES

1. Ducatel, K., M. Bogdaniwicz, F. Scapolo, J. Leijten and J. Burgelman, *Scenarios for Ambient Intelligence in 2010*, 2001, `http://fiste.jrc.ec.europa.eu/pages/detail.cfm?prs=587`, January 2011.

2. Shadbolt, N., "Ambient Intelligence", *IEEE Intelligent Systems*, 2003.

3. Aarts, E., R. Harwig and M. Schuurmans, *The Invisible Future: The Seamless Integration of Technology into Everyday Life*, chap. Ambient Intelligence, pp. 235–250, McGraw-Hill, Inc., New York, USA, 2001.

4. Aarts, E., "Ambient Intelligence: A Multimedia Perspective", *IEEE Multimedia*, Vol. 11, pp. 12–19, 2004.

5. Austin, J. L., *How to Do Things with Words*, Clarendon Press, Oxford, 1962.

6. WP12, *D12.2: Study on Emerging AmI Technologies*, Future of Identity in the Information Society Consortium, October 2007, `www.fidis.net`, May 2011.

7. Jennings, N. R., "Agent-based Computing: Promise and Perils", *International Joint Conference on Artificial Intelligence*, Vol. 16, pp. 1429–1436, 1999.

8. Wooldridge, M. J., *An Introduction to Multiagent Systems*, Wiley, 2002.

9. Weiss, G. (Editor), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge, MA, USA, 1999.

10. Winer, M. and K. Ray, *Collaboration Handbook: Creating, Sustaining, and Enjoying the Journey*, Fieldstone Alliance, 1994.

11. Singh, M. P., "An Ontology for Commitments in Multiagent Systems", *Artificial*

Intelligence and Law, Vol. 7, No. 1, pp. 97–113, 1999.

12. Walton, D. and E. C. W. Krabbe, *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*, SUNY Press, August 1995.

13. Castelfranchi, C., "Commitments: From Individual Intentions to Groups and Organizations", *Proceedings of the First International Conference on Multi-Agent Systems*, pp. 41–48, 1995.

14. Saffer, D., *Designing for Interaction:Creating Smart Applications and Clever Devices*, Peachpit Press, 2006.

15. Sadri, F., "Multi-Agent Ambient Intelligence for Elderly Care and Assistance", *Proceedings of the American Institute of Physics Conference*, pp. 117–120, 2007.

16. Cabri, G., L. Ferrari, L. Leonardi and F. Zambonelli, "The LAICA Project: Supporting Ambient Intelligence via Agents and Ad-Hoc Middleware", *14th IEEE International Workshops on Enabling Technologies*, pp. 39–46, 2005.

17. Costantini, S., L. Mostarda, A. Tocchio and P. Tsintza, "DALICA: Agent-Based Ambient Intelligence for Cultural-Heritage Scenarios", *IEEE Intelligent Systems*, Vol. 23, No. 2, pp. 34–41, Mar. 2008.

18. Moraitis, P. and N. Spanoudakis, "Argumentation-based Agent Interaction in an Ambient-intelligence Context", *IEEE Intelligent Systems*, pp. 84–93, 2007.

19. Holmes, A., H. Duman and A. Pounds-Cornish, "The iDorm: Gateway to Heterogeneous Networking Environments", *International ITEA Workshop on Virtual Home Environments*, pp. 20–21, Paderborn, Germany, 2002.

20. Chen, H., T. Finin and A. Joshi, "Intelligent Agents Meet the Semantic Web in Smart Spaces", *IEEE Internet Computing*, pp. 69–79, 2004.

21. Desai, N., A. K. Chopra and M. P. Singh, "Representing and Reasoning About

Commitments in Business Processes", *Proceedings of the National Conference on Artificial Intelligence*, pp. 1328–1333, 2007.

22. Fornara, N. and M. Colombetti, "Ontology and Time Evolution of Obligations and Prohibitions Using Semantic Web Technology", *Declarative Agent Languages and Technologies VII*, pp. 101–118, 2010.

23. Gruber, T., "Toward Principles for the Design of Ontologies Used for Knowledge Sharing", *International Journal of Human-Computer Studies*, Vol. 43, No. 5-6, pp. 907–928, Nov. 1995.

24. Genesereth, M. R. and N. J. Nilsson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, 1987.

25. Gruber, T., "A Translation Approach to Portable Ontology Specifications", *Knowledge Creation Diffusion Utilization*, Vol. 5, No. April, pp. 199–220, 1993.

26. Enderton, H. B., "Degrees of Computational Complexity", *Journal of Computer and System Sciences*, Vol. 6, No. 5, pp. 389–396, 1972.

27. Staab, S. and R. Studer (Editors), *Handbook on Ontologies*, International Handbooks on Information Systems, Springer, 2004.

28. HP Labs, *Jena - A Semantic Web Framework for Java*, 2002, `http://jena.sourceforge.net/index.html`, May 2011.

29. Bellifemine, F., A. Poggi and G. Rimassa, "JADE–A FIPA-compliant agent framework", *Proceedings of the Fourth International Conference on the Practical Application of Intelligent Agents and Multi-agent Technology*, Vol. 99, pp. 97–108, 1999.

30. Howden, N., R. Rönnquist, A. Hodgson and A. Lucas, "JACK Intelligent Agents–Summary of an Agent Infrastructure", *5th International Conference on Autonomous Agents*, 2001.

31. Bordini, R. H., J. F. Hübner and M. J. Wooldridge, *Programming Multi-agent Systems in AgentSpeak Using JASON*, Wiley, 2007.

32. Rao, A. S., "AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language", *Modelling Autonomous Agents in a Multi-Agent World*, pp. 42–55, 1996.

33. W3C OWL Working Group, *OWL 2 Web Ontology Language: Document Overview*, W3C Recommendation, 2009, `http://www.w3.org/TR/owl2-overview/`, may 2011.

34. Chen, H., T. Finin and A. Joshi, "Semantic Web in the Context Broker Architecture", *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications*, pp. 277–286, 2004.

35. Chen, H., F. Perich, T. Finin and A. Joshi, "SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications", *International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pp. 258–267, IEEE Computer Society, Boston, MA, August 2004.

36. Sadeh, N., F. Gandon and O. Kwon, *Ambient intelligence: The MyCampus Experience*, 2005, `http://handle.dtic.mil/100.2/ADA482307`, April 2011.

37. Ankolekar, A., M. Burstein, J. R. Hobbs, O. Lassila and D. L. M. et. al. et. al., "DAML-S: Web Service Description for the Semantic Web", *International Semantic Web Conference*, pp. 348–363, Springer, 2002.

38. Moraitis, P., E. Petraki and N. I. Spanoudakis, "An Agent-Based System for Infomobility Services", *European Workshop on Multi-Agent Systems*, pp. 224–235, Brussels, 2005.

39. Poslad, S., H. Laamanen, R. Malaka, A. Nick, P. Buckle and A. Zipl, "Crumpet: Creation of User-friendly Mobile Services Personalised for Tourism", *Second International Conference on 3G Mobile Communication Technologies*, pp. 28–32,

IET, 2001.

40. Foundation for Intelligent Physical Agents, *FIPA Personal Travel Assistance Specification*, 2001, `http://www.fipa.org/specs/fipa00080/XC00080B.html`, May 2011.

41. Open Service Gateway Initiative, *OSGI*, 2011, `http://www.osgi.org`, May 2011.

42. Costantini, S. and A. Tocchio, "The DALI Logic Programming Agent-oriented Language", *Logics in Artificial Intelligence*, pp. 685–688, 2004.

43. Satoh, I., "Mobile Agents for Ambient Intelligence", *Massively Multi-Agent Systems*, pp. 187–201, 2005.

44. Lech, T. and L. Wienhofen, "AmbieAgents: A Scalable Infrastructure for Mobile and Context-aware Information Services", *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 625–631, ACM, 2005.

45. Göker, A., S. Watt, H. I. Myrhaug, N. Whitehead, M. Yakici, R. Bierig, S. K. Nuti and H. Cumming, "An Ambient, Personalised, and Context-sensitive Information System for Mobile Users", *Proceedings of the Second European Union Symposium on Ambient Intelligence*, pp. 19–24, ACM Press, New York, November 2004.

46. Seghrouchni, A. E. F., K. Breitman, N. Sabouret, M. Endler, Y. Charif and J.-P. Briot, "Ambient Intelligence Applications: Introducing the Campus Framework", *Proceedings of the 13th IEEE International Conference on Engineering of Complex Computer Systems*, pp. 165–174, IEEE, March 2008.

47. Sacramento, V., M. Endler, H. K. Rubinsztejn, L. S. Lima, K. Goncalves, F. N. Nascimento and G. A. Bueno, "MoCA: A Middleware for Developing Collaborative Applications for Mobile Users", *IEEE Distributed Systems Online*, Vol. 5, No. 10, p. 2, October 2004.

48. Hellenschmidt, M. and T. Kirste, "SodaPop: A Software Infrastructure Supporting Self-organization in Intelligent Environments", *Second IEEE International Conference on Industrial Informatics*, pp. 479–486, IEEE, 2004.

49. Işıksal, A., *Creating and Enacting Dynamic Contracts for Ambient Intelligence*, Master's Thesis, Bogazici University, Istanbul, June 2011.

50. Mengistu, D., P. Tröger, L. Lundberg and P. Davidsson, "Scalability in Distributed Multi-Agent Based Simulations: The JADE Case", *The Second International Conference on Future Generation Communication and Networking Symposia*, pp. 93–99, December 2008.