# Contract-Based Cooperation for Ambient Intelligence: Proposing, Entering and Executing Contracts Autonomously

Fatma Başak Aydemir[*]
Dept of Computer Engineering
Boğaziçi University
İstanbul, Turkey
basak.aydemir@boun.edu.tr

## ABSTRACT

Ambient Intelligence (AmI) describes environments that sense and react to the humans in time to improve their living quality. Software agents are important in realizing such environments. While existing work has focused on individual agent's reactions, more interesting applications will take place when the agents cooperate to provide composed services to humans. When cooperation is required, the environment needs mechanisms that regulate the agents' interactions but also respect their autonomy. Accordingly, this paper develops a contract-based approach for offering composed services. At runtime, agents autonomously decide whether they want to enter contracts. Agents then act to fulfill their contracts. Ontologies are used to capture domain information. 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.

## Keywords

Agents, commitments, ontologies

## 1. INTRODUCTION

Ambient Intelligence (AmI) indicates environments that are aware of and responsive to human presence. Besides various types of sensors and nanotechnology, software agents are one of the emerging technologies for AmI. Intelligent agents are used for a wide range of tasks from searching for information to adaptive decision making [11]. With this aspect of it, AmI can be realized by a multiagent system. Multiagent systems are systems where multiple intelligent agents interact [10]. These interactions are generally given a meaning using commitments, which are contracts among agents to satisfy certain properties [8]. Using contracts among agents regulate the interactions and enable cooperation among them.

In this paper, we propose an AmI system which consists of autonomous 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 [7] for this system, 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). Other agents cooperate with UA in order to satisfy the user's needs. One distinguishing aspect is that predefined contracts, which are generated before agent interaction, do not exist in the system. 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. 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. When a contract cannot be created, it is UA's duty to establish another one that guarantees realization of the properties needed to satisfy the user.

The rest of the paper is organized as follows: Section 2 explains the advantages of the dynamically generated contracts over the statically generated ones. Section 3 describes the overall system architecture and explains the contract evolutions. Section 4 demonstrates the application of the system on an example domain. Section 5 studies the system over selected scenarios and Section 6 compares the system with the related work.

## 2. CONTRACTS FOR AMBIENT INTELLIGENCE

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. In some multiagent systems, the system is designed so that the role of the agents are set, agent capabilities do not change, the resources to realize these capabilities are determined and the agents' access to these resources are unlimited. In such static environments, contracts can be specified during compile time and agents can follow these contracts at run time. Since the system is not going to change at run time, there is no reason to attempt to generate the contracts at run time.

Consider a multiagent AmI system with UA and two other agents, Agent 1 and Agent 2. Assume that the following contracts are generated at design time and adopted by the agents:

[*]This paper is based on the STAMI 2011 workshop paper [2] at IJCAI. The author is supported by Boğaziçi University Research Fund under grant BAP5694, by the Scientific and Technological Research Council of Turkey (TÜBİTAK) 2210 National Graduate Scholarship Program.

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

That is, if UA requests Service 1, then Agent 1 will always provide that service. Similarly, if UA requests Service 2, then Agent 2 will always provide that service. These two contracts work well as long as the agents of the system, their capabilities, the resources and the user preferences do not change.

**Scarce Resources:** The scenario depicted above is far from being realistic. Any change in the environment prevents the system from satisfying the user's needs. Consider the case that the resources necessary to provide the services 1 and 2 are not 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. This leads to an overall system failure since UA is not served a part of the service bundle. In such cases, the statically generated contracts described above are not sufficient to realize the user's preferences. Instead, the agents should decide whether or not to take part in the contracts and also they should try to generate new contracts that may help to fulfill the former ones. For the Scarce Resource 1 example, Agent 1 may ask for a new contract including the following commitment: CC(Agent 1, UA , Resource 1, Service 1), which means that if UA provides Resource 1, then Agent 1 can provide Service 1. If UA accepts the new proposed contract and provides Resource 1 to Agent 1, Agent 1 provides Service 1 to UA . Service 1 would not be provided if the later contract had not been generated by Agent 1 dynamically.

**Dynamic 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. When UA tries to serve a bundle, states of the agents should be considered. It is not rational to wait for a service from an agent that has already left the system, although it is committed to serve it. So, the appropriate agents should be carefully selected before agreeing on any commitment. For example, in the above scenario, Agent 1 decides to leave the system for some reason, meanwhile, a new agent, Agent 3, which offers the same services as Agent 1 enters the system. Although there is a contract agreed on with Agent 1, in order to receive Service 1, UA should make another contract with Agent 3: CC(UA , Agent 3, Service 1 Request, Service 1). If UA can dynamically create a new contract with Agent 3, it can ensure receiving Service 1.

**Dynamically Changing Services:** A multiagent system does not necessarily contain agents that have fixed services. Agents may learn new services or stop some of the existing ones. In such cases, making prior arrangements to serve a bundle may not work due to change of agent services. The agents to be interacted with should be carefully selected according to the services they offer. In such systems, it may also be impossible to serve a predetermined bundle, the service bundle may be generated dynamically too.

## 3. APPROACH

We develop a contract-based multiagent system for ambient intelligence. The agents cooperate by creating and carrying out contracts that they dynamically generate at run time.

**Architecture:** Main components of the system are depicted in Figure 1. Agents are shown in rectangle nodes and the ontologies are shown in ellipse nodes. Line edges describe two way interaction whereas dashed edges represent access to the ontologies.

There is one UA which 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. Elements of this set are often served by various agents, so other agents cooperate with UA to offer their services. UA usually starts the communication, however other agents are also able to make contract requests. All agents make the decision for whether or not entering in a contract themselves.

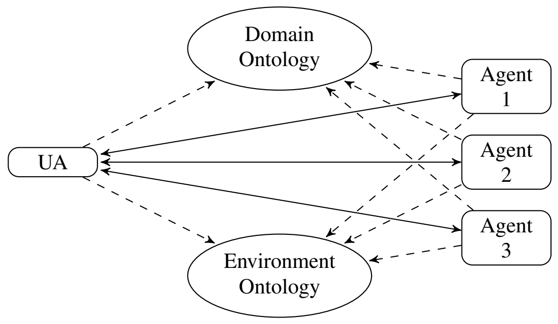

**Figure 1: Architecture of the system**

There are two ontologies that are accessible by all of the agents in the system. An ontology is the description of the conceptualization of a domain [1]. Elements that are described in an ontology are the individuals, that are the objects of the domain; classes, that are collections of objects; attributes, that are properties of objects; relations that are the connections between objects and rules defined on these elements [5].

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 initial state and managed by the related agent itself. The second ontology is the domain ontology. In this ontology, detailed descriptions of the services and the other domain dependent information are provided.

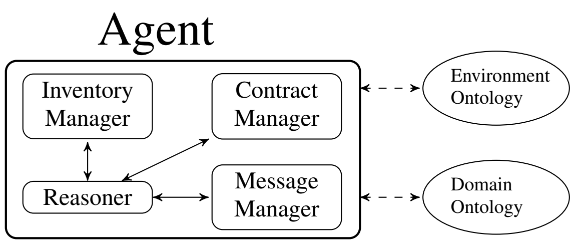

**Figure 2: Architecture of an agent**

Figure 2 depicts the structure of an agent. Each agent in the system has access to the environment ontology and the domain ontology. Every agent has a local inventory where it keeps the availability information on the service resources. The inventory of an agent is consulted first to decide if the necessary service resources are available. The information about the agent's inventory is private and it is not shared with the other agents of the system. 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. Obviously, 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 reasoner of the agent makes the decisions, takes actions and handles messages. **Contract Lifecycle:**

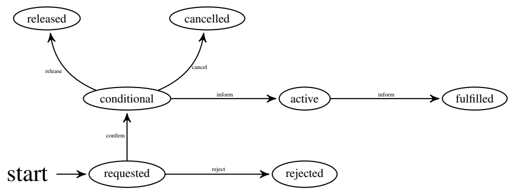

**Figure 3: Commitment states as nodes and message types as edges**

In our system, the interaction among agents is conducted via messages and it is based on 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 released.
- 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).

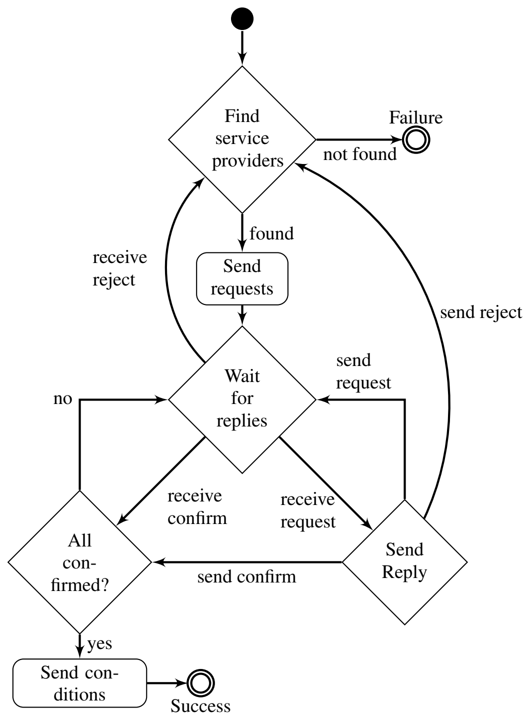

**Figure 4: Workflow of User Agent**

Figure 3 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 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.

**Agent Lifecycle:** Workflow diagram for UA is given in Fig. 4. When UA tries to establish the contracts for a service bundle, it starts with getting the addresses of the agents that provides the services from the bundle. If it cannot find any agents for one or more services, the bundle cannot be served (Failure). If there are agents that serve the services of the bundle, UA sends them contract 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. 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. 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 no 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

**Algorithm 1:** Request Received

```
   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         found = true;
12         break;
13 if !found then
14     ResourceList rList=c.getProposition();
15     ResourceList missing;
16     for i ← 1 to gList.size() do
17         Resource r=rList.elementAt(i);
18         double invQ=Inventory.getResourceQuantity(r);
19         if g.RequestedQuantity > invQ then
20             missingResources(g,missing);
21             if missing.size()!=0 then
22                 m.type ← request;
23                 c.condition ← missing;
24                 m.add(c);
25                 return m
26         m.type ← confirm;
27         m.add(c);
28         return m
```

are willing to create contracts unless they lack necessary amount of ingredients and they do not receive any contract requests beyond their serving capabilities.

Algorithm 1 explains the behavior of an agent other than UA when it receives a request message. The message received can start a new conversation between the 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).

In addition to receiving a 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, it updates the state to active. This means that, the agent itself is now responsible to carry out the rest of the contract by bringing about its proposition. On the other hand, if it finds a contract whose proposition matches the content and its state is active, meaning if the sender agent is fulfilling a contract, it updates its state to fulfilled.

## 4. EXAMPLE DOMAIN

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. The 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 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.

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. 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 substitude 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.

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.

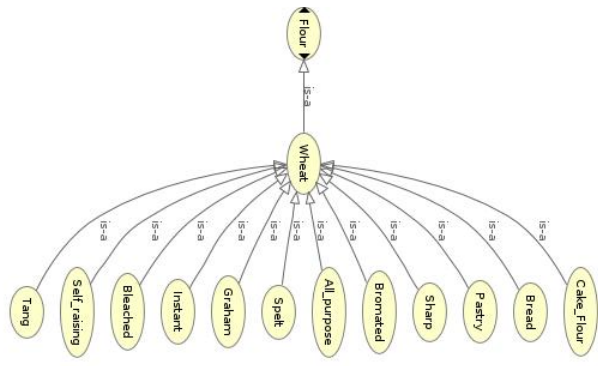

**Figure 5: A caption from example domain ontology, representing flour class**

## 4.1 Scenario 1

For the first scenario, user tells UA that she wants a menu consisting of two different services, coffee and cake, which should be served together. UA needs to find the agents serving the menu items, for this case they are CMA and FA . Then, UA needs to establish contracts for all of the items in the menu and receive the items. CMA needs some coffee beans to serve coffee and it manages to create a contract when it accepts to provide coffee beans to CMA . Once all contracts are established, UA fulfills the conditions of the contracts and gets served.

## 4.2 Scenario 2

For the second scenario, UA again tries to serve a coffee and cake menu of the user's choice. The menu item coffee is served by CMA and the cake is served by MA. UA establishes a contract with CMA. However, MA is out of cake flour which is essential for serving a cake. It requests some from UA , however UA cannot provide it and after consulting the domain ontology, UA offers bread flour, which is a replacement for the original ingredient. Once again, after all contracts are established, UA fulfills the conditions of the contracts and gets served.

## 4.3 Scenario 3

The third scenario begins similar to the second one. UA tries to establish contracts for the coffee and cake menu. It establishes one with CMA . MA asks for a substitute for the cake flour, which is an ingredient to make the cake. Not being able to provide the cake flour, UA offers bread flour. However, this time MA does not find the substitute similar enough to replace the original item. UA cannot establish a contract with Mixer Agent and looks for another agent that can provide cake. It discovers FA and establishes a contract with it. UA fulfills the conditions and waits for the services but CMA gets broken and does not respond.

## 5. RESULTS

JADE [3] agent development framework is used to implement the agents, which natively provides the messaging system, the yellow pages and the distributed system architecture. The yellow pages service is given by the Directory Facilitator (DF) Agent of each container and once the agents register their services to the DF, others can find them through a query to the DF. The agent implementation is separated from the underlying details of the messaging service.

## 5.1 Execution of Scenario 1

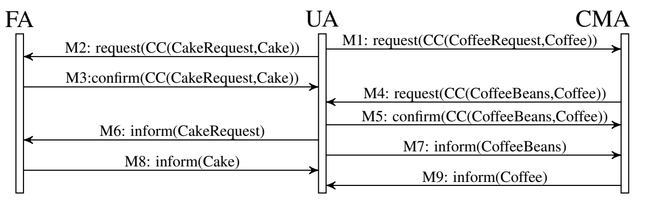
FA
UA
CMA
M1: request(CC(CoffeeRequest,Coffee))
M2: request(CC(CakeRequest,Cake))
M3:confirm(CC(CakeRequest,Cake))
M4: request(CC(CoffeeBeans,Coffee))
M5: confirm(CC(CoffeeBeans,Coffee))
M6: inform(CakeRequest)
M7: inform(CoffeeBeans)
M8: inform(Cake)
M9: inform(Coffee)

**Figure 6: Sequence Diagram for Scenario 1**

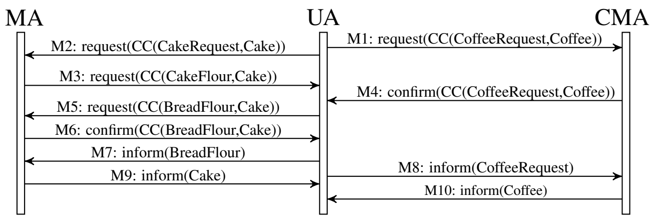
MA
UA
CMA
M1: request(CC(CoffeeRequest,Coffee))
M2: request(CC(CakeRequest,Cake))
M3: request(CC(CakeFlour,Cake))
M4: confirm(CC(CoffeeRequest,Coffee))
M5: request(CC(BreadFlour,Cake))
M6: confirm(CC(BreadFlour,Cake))
M7: inform(BreadFlour)
M8: inform(CoffeeRequest)
M9: inform(Cake)
M10: inform(Coffee)

**Figure 7: Sequence Diagram for Scenario 2**

Figure 6 depicts the scenario described in section 4.1. 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). FA immediately sends a confirmation back (M 3). On the other hand CMA is in need of some coffee beans, so it sends a request message back (M 4). UA accepts this offer (M 5). 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). 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).

## 5.2 Execution of Scenario 2

For the scenario described in Section 4.2, the flow of communication is depicted in Fig. 7. UA sends relevant request messages to start conversation (M 1 and M 2). 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). 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). The substitute satisfies MA and it accepts to take part in the contract (M 6). 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). After receiving the conditions, agents serve the propositions of their contracts (M 9, 10).

## 5.3 Execution of Scenario 3

The communication flow between agents for the scenario described in Section 4.3 is represented in Fig. 8. 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 confirma-

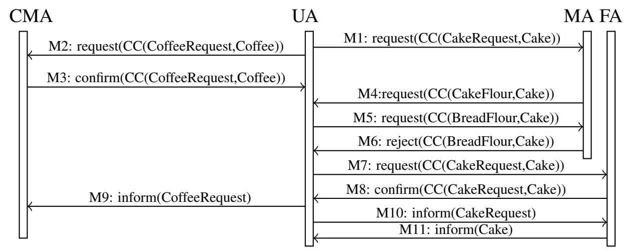
CMA
UA
MA FA
M1: request(CC(CakeRequest,Cake))
M2: request(CC(CoffeeRequest,Coffee))
M3: confirm(CC(CoffeeRequest,Coffee))
M4:request(CC(CakeFlour,Cake))
M5: request(CC(BreadFlour,Cake))
M6: reject(CC(BreadFlour,Cake))
M7: request(CC(CakeRequest,Cake))
M8: confirm(CC(CakeRequest,Cake))
M9: inform(CoffeeRequest)
M10: inform(CakeRequest)
M11: inform(Cake)

**Figure 8: Sequence Diagram for Scenario 3**

tion, 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.

## 6. DISCUSSION

The main contribution of our work is to dynamically generate and use contracts to ensure that the user's needs are satisfied in a dynamic environment. Unlike Hagras *et al.*, we do not assume that any agent serving a person must always and immediately carry out any requested actions [6]. Instead, we develop a model for an open dynamic system where the continuity of the services are secured, even when some agents stop working or leave the system, without being noticed by the user.

Although it is assumed that the agents are willing to cooperate under certain conditions in Section 5, the model which is represented in this paper does not have a predefined communication protocol. The existence of less or more cooperative agents in the system does not destroy the system's ability to operate. We can also say that the agents in the system do not have designated roles, as they can change the services provided by them.

We benefit from the ontologies to achieve a high degree of interoperability; however, the contracts that are generated in the system are not kept in ontologies like in the case of Fornara and Colombetti [4]. Since the evolution of the contracts are handled by the agents, our model deliberately lacks a central monitoring system, which has access the information on all of the transactions. Hence, contracts are also kept independently.

Unlike some AmI frameworks such as Amigo [9], our application does not offer a low level interoperation structure. In Amigo framework agents do not have any options but to provide their services when their relevant methods are called by the other agents. Also, in that framework, exact structure of the service methods of the provider agent such as the parameters and the name and so on should be known by the demanding agent. Instead of such framework, we provide a high level interaction model where agents willingly provide their services or not. It is not necessary for the demanding agent to know the details about the provider agent's methods.

Future work may include the development of a policy for exceptions. The sanctions that will be applied to an agent that does not follow a contract should be set to avoid the abuse of the system. Also, the cancellation and release policies for agents should be defined, so that the agents can inform the other party when they cannot deliver the services they are committed to.

## 7. REFERENCES

[1] E. Aarts. Ambient intelligence: A multimedia perspective. *IEEE Multimedia*, 11:12–19, 2004.

[2] F. B. Aydemir and P. Yolum. Contract-based cooperation for ambient intelligence. In *The Workshop on Space Time and Ambient Intelligence*, Barcelona, July 2011. accepted.

[3] F. Bellifemine, A. Poggi, and G. Rimassa. JADE–A FIPA-compliant agent framework. In *Proceedings of PAAM*, volume 99, pages 97–108, 1999.

[4] N. Fornara and M. Colombetti. Ontology and time evolution of obligations and prohibitions using semantic web technology. *Declarative Agent Languages and Technologies VII*, pages 101–118, 2010.

[5] T. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5-6):907–928, Nov. 1995.

[6] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, and H. Duman. Creating an ambient-intelligence environment using embedded agents. *Intelligent Systems, IEEE*, 19(6):12–20, 2004.

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

[8] M. P. Singh. An ontology for commitments in multiagent systems. *Artificial Intelligence and Law*, 7(1):97–113, 1999.

[9] G. Thomson, D. Sacchetti, Y.-D. Bromberg, J. Parra, N. Georgantas, and V. Issarny. Amigo Interoperability Framework: Dynamically Integrating Heterogeneous Devices and Services. *Constructing Ambient Intelligence*, pages 421–425, 2008.

[10] M. J. Wooldridge. *An introduction to multiagent systems*. Wiley, 2002.

[11] WP12. *D12.2: Study on Emerging AmI Technologies*. Future of Identity in the Information Society Consortium, www.fidis.net., October 2007.