UNIVERSITY OF CRETE – HERAKLION GREECE
DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF SCIENCES AND ENGINEERING

Multi-Agent Distributed Epistemic Reasoning in Ambient Intelligence Environments

Master Thesis

Hatzivasilis Georgios

November 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Κατανεμημένη Επιστημολογική Συλλογιστική με Πολλαπλούς Πράκτορες σε Περιβάλλοντα Διάχυτης Νοημοσύνης

Εργασία που υποβλήθηκε από τον
Χατζηβασίλη Γεώργιο
ως μερική εκπλήρωση των απαιτήσεων για απόκτηση
ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

Χατζηβασίλης Γεώργιος, Πανεπιστήμιο Κρήτης

Εισηγητική Επιτροπή:

Δημήτρης Πλεξουσάκης, Καθηγητής,
Πανεπιστήμιο Κρήτης, Πρόεδρος

Γρηγόρης Αντωνίου, Καθηγητής,
Πανεπιστήμιο Κρήτης, Μέλος

Ιωάννης Τσαμαρδίνος, Επίκουρος Καθηγητής,
Πανεπιστήμιο Κρήτης, Μέλος

Δεκτή από:

Άγγελος Μπίλας, Αναπληρωτής Καθηγητής,
Πανεπιστήμιο Κρήτης

Πρόεδρος επιτροπής Μεταπτυχιακών Σπουδών
Ηράκλειο, Νοέμβριος 2011

Multi-Agent Distributed Epistemic Reasoning in Ambient Intelligence Environments

ABSTRACT

In Ambient Intelligence environments, there exist many different entities, called agents, which collect, process and exchange information about these environments. All agents coexist in the same environment and share the same context. However, each agent faces it from a different aspect according to its role, capabilities, authority and goals. Every agent comes up with its own viewpoint about the environment, acting as an individual entity but cooperating with other agents to accomplish its goals and thus forming an agent system or network. The global consistency of the whole system has introduced new research challenges in the Ambient Intelligence (AmI) field. As agents are sensing the environment variables, incorrect information can arise from missing facts and ambiguous information among the different agents' perceptions.

In this thesis, we model agents as nodes in a peer-to-peer network, considering the conflicts that may arise during the integration of the knowledge distribution. We propose a proof of evidence mechanism to resolve the situations that may arise which is based on a grading mechanism, called certainty degree, and on share theories, which are the combined sub-theories of the participating agents. We examine consistency matters about both the individual theories and the share theories that may be constructed.

When we implement an intelligent system we must be able to model several cases of real world states and problems. We present a process of real time distributed reasoning for ambient environments. We use the Event Calculus (EC) as a logic language to model the AMI environment, the agents' theories about these environments and the events that can occur. We have extended the basic reasoning process of EC and resolve problems where reasoning about real system time must be considered, like "Turn off the oven in 20 minutes" or "if somebody enters the room in the next 5 minutes, send me a message". Thus, problems like the conditional 'n>m' or the 'nth occurrence' can be encountered in the real time space. We even enrich the expressiveness of our tool by enabling the modeling of contexts with knowledge, preferences and priorities.

Κατανεμημένη Επιστημολογική Συλλογιστική με Πολλαπλούς Πράκτορες σε Περιβάλλοντα Διάχυτης Νοημοσύνης

ΠΕΡΙΛΗΨΗ

Στα περιβάλλοντα Διάχυτης Νοημοσύνης, υπάρχουν πολλές διαφορετικές οντότητες, οι πράκτορες, οι οποίες συλλέγουν, επεξεργάζονται και ανταλλάσουν πληροφορίες για αυτά τα περιβάλλοντα. Όλοι οι πράκτορες συνυπάρχουν στο ίδιο περιβάλλον και μοιράζονται το ίδιο πλαίσιο. Ωστόσο, κάθε πράκτορας το αντιμετωπίζει από μία διαφορετική οπτική γωνία σύμφωνα με τον ρόλο, τις δυνατότητες, τα δικαιώματα χρήσης και τους στόχους του. Κάθε πράκτορας δημιουργεί την δική του αντίληψη για το περιβάλλον, δρώντας ως αυτόνομη οντότητα αλλά συνεργαζόμενη με τους υπόλοιπους πράκτορες προκειμένου να εκπληρώσει τους στόχους του και σχηματίζοντας έτσι ένα σύστημα ή δίκτυο πρακτόρων. Η καθολική συνέπεια του όλου συστήματος έχει εισάγει νέες ερευνητικές προκλήσεις στον τομέα της Διάχυτης Νοημοσύνης. Καθώς οι πράκτορες διαισθάνονται τις παραμέτρους του περιβάλλοντος, λανθάνουσες πληροφορίες μπορούν να προκύψουν από ελλιπή γεγονότα και αμφίσημες πληροφορίες μεταξύ των διαφορετικών αντιλήψεων των πρακτόρων.

Σε αυτή την διατριβή, μοντελοποιούμε τους πράκτορες ως κόμβους σε ένα δίκτυο peer-to-peer, λαμβάνοντας υπόψη συγκρούσεις που μπορούν να προκύψουν κατά την διάρκεια της ενοποίησης της κατανεμημένης γνώσης. Προτείνουμε έναν μηχανισμό για την απόδειξη στοιχείων (proof of evidence mechanism) για την επίλυση των υποθέσεων που μπορούν να ανακύψουν ο οποίος βασίζεται σε έναν μηχανισμό βαθμολόγησης, τον βαθμό βεβαιότητας, και στις διαμοιραζόμενες θεωρίες, οι οποίες είναι οι συγχωνευμένες υπο-θεωρίες των πρακτόρων που συμμετέχουν σε μία διαμάχη. Εξετάζουμε θέματα συνέπειας και για τις αυτόνομες θεωρίες και για τις διαμοιραζόμενες θεωρίες που μπορεί να κατασκευαστούν.

Όταν υλοποιούμε ένα ευφυές σύστημα πρέπει να είμαστε σε θέση να μοντελοποιήσουμε αρκετές περιπτώσεις καταστάσεων και προβλημάτων του πραγματικού κόσμου. Παρουσιάζουμε μία διαδικασία για κατανεμημένη συλλογιστική πραγματικού χρόνου για περιβάλλοντα Διάχυτης Νοημοσύνης. Χρησιμοποιούμε τον Λογισμό Συμβάντων (Event Calculus, EC) ως την γλώσσα λογικού προγραμματισμού για να μοντελοποιήσουμε τα περιβάλλοντα Διάχυτης Νοημοσύνης, τις λογικές θεωρίες των πρακτόρων για αυτά τα περιβάλλοντα και τα γεγονότα που μπορούν να συμβούν. Έχουμε επεκτείνει την βασική συλλογιστική διαδικασία του Λογισμού Συμβάντων και να επιλύουμε προβλήματα όπου συλλογιστική με τον πραγματικό χρόνο του συστήματος πρέπει να μελετηθεί, όπως "Σβήσε τον φούρνο σε 20 λεπτά" ή "εάν κάποιος μπει στο δωμάτιο στα επόμενα 5 λεπτά, στείλε μου ένα μήνυμα". Έτσι, προβλήματα όπως η υπό συνθήκη 'n>m' ή η 'n$^{\text{στη}}$ εμφάνιση' μπορούν να αντιμετωπιστούν σε πραγματικό χρόνο. Εμπλουτίζουμε ακόμα περισσότερο την εκφραστικότητα του εργαλείου μας δίνοντας την δυνατότητα μοντελοποίησης πλαισίων με γνώση, προτιμήσεις και προτεραιότητες.

Επόπτης Καθηγητής:

Δημήτρης Πλεξουσάκης

Καθηγητής Τμήματος Επιστήμης Υπολογιστών

Πανεπιστήμιο Κρήτης

Στους γονείς μου, Ελένη και Βασίλη, και τον αδερφό μου Χαράλαμπο για την αγάπη και την υποστήριξή τους.

Ευχαριστίες

Αισθάνομαι την ανάγκη να ευχαριστήσω τον επόπτη καθηγητή μου, κύριο Δημήτρη Πλεξουσάκη για την πολύτιμη καθοδήγηση αλλά και για τις ουσιαστικές συμβουλές του με τις οποίες συνέβαλλε στην επιτυχή ολοκλήρωση των μεταπτυχιακών μου σπουδών. Θα ήθελα να ευχαριστήσω τον καθηγητή κύριο Γρηγόρη Αντωνίου για τις εύστοχες παρατηρήσεις και συμβουλές του ως μέλος της εισηγητικής επιτροπής της εργασίας όπως επίσης και τον επίκουρο καθηγητή κύριο Ιωάννη Τζαμαρδίνο για τη συμμετοχή του στην εισηγητική επιτροπή.

Αυτή η εργασία υποστηρίχθηκε με χορήγηση μεταπτυχιακής υποτροφίας από τον Ινστιτούτο Πληροφορικής (ΙΠ) του Ιδρύματος Έρευνας και Τεχνολογίας (ΙΤΕ). Επιπλέον, θα ήθελα να ευχαριστήσω για την βοήθεια και την υποστήριξη που μου πρόσφεραν η γραμματεία και το τεχνικό προσωπικό του τμήματος Επιστήμης Υπολογιστών του Πανεπιστημίου Κρήτης και του ΙΤΕ.

List of Figures

List of Tables

Table of Contents

1. Introduction

The knowledge distribution remains an open issue in the field of Distributed Artificial Intelligence. This is mainly caused by the imperfect nature of the available context information and the special characteristics of the agents that provide and process this knowledge. An agent is likely to require pieces of information about the environment that are not accessibly by him. Thus, agents must join in a mechanism where each agent would be able to search for those missing pieces of information and get informed about future changes that could occur. Bikakis in [1] denotes three main challenges of knowledge management in AmI:

1. Reasoning with the highly dynamic and ambiguous context data
2. Managing the potentially huge piece of context data, in a real-time fashion, considering the restricted computational capabilities of some mobile devices
3. Collective intelligence, by supporting information sharing and distributed reasoning between the entities of the ambient environment

Distributing the intelligence makes the system easier to adapt and more scalable. AmI systems must provide the ability in different devices to dynamic join and leave the environment and also the flexibility of adding extra functionality to serve the future user requirements. Distributed approaches can meet the AmI requirements.

Centralized approaches can achieve better control and coordination between the system's components. However, these approaches are about to fail in satisfying the needs of AmI systems because they often imply risks that are not feasible for these environments. There might be a lack of processing power, memory bottlenecks or a loss of centralized data that may affect the performance of the system. Moreover, a failure in the centralization component signifies the failure of the whole system and the unavailability of the service.

There are also other reasons why distributed solutions may be necessary, analyzed in [2], like:

- cost of formalization
- privacy
- dynamicity
- brittleness

This study proposes a distributed solution for a multi-agent system which is adaptable to AmI environments and fulfills their demands. We choose the multi-agent model because it is more suitable to represent an AmI environment as it comprises the advantages enumerated in [3] such as:

- Monitoring and control versatility
- Better resource allocation

- Facilitate system design
- Allow modularity and flexibility
- Robust behavior on automated processes

We adopt the *super node* approach from the peer-to-peer networking. A super node is a node that also serves as one of that network's relayers, handling data flow, connections for other users and keeps an indexing mechanism of what information it serves. Ordinarily, a super node would be a node with better features than the average node. These kinds of networks were designed to allow nodes leaving and entering the network dynamically. When a node enters in the network, it is connecting with a super node. The super node keeps track of the information that it serves for each of these nodes. If the payload for a super node is increased, another super node can undertake the excess burden. As the network is becoming larger more nodes are dynamically transformed to super nodes and as the network is becoming smaller the number of super nodes decreases. When a super node must be turned off or it must stop acting as a super node, it splits its indexing information and shares it to the remaining super nodes. Systems like Skype and other systems that support file sharing, use this topology. This semi-distributed architecture allows data to be decentralized without requiring excessive overhead at every node. Moreover this topology enables us to search information that is available from the nodes and the super nodes.

In our implementation, we consider the agents as nodes in such a peer-to-peer network. Each node acts as an individual entity, performing its own actions, like sensing and reasoning, and maintaining its own local knowledge base. The agents are interacting with each other through the network by exchanging that knowledge. The internal knowledge is formed by rules, facts and constraints and the external knowledge from other agents is distributed as pieces of information which can be sensed.

There are two basic types of agents: Simple Agents (SAs) or agents & Master Agents (MAs). Simple agents are the majority in our system, which usually have lower capabilities for storing data and CPU. The simple agents are sensing local events, maintain their own knowledge base, perform their own reasoning process and distribute their knowledge about their environment variables to the agent system. A simple agent can also maintain locally protected knowledge, which is not distributed to the system. Each simple agent is connected to the nearest master agent, to whom the distributed knowledge is sent. Ordinarily, there is a simple agent for each smart device in our environment.

The master agents are the backbone of the system. They adopt the functionality of a super node. These agents support the system full communication among all agents and are responsible for the knowledge distribution. The master agents receive the local knowledge from the simple agents that are connected to them and publish it. In this version of the system there are not supported access policies for the distributed data. Each piece of information that a simple agent chooses to publish to the system, is globally available. Each master agent can communicate with the simple agents that are attached to it, and with its nearest master agents. A master agent could also control local variables. Specifically, a master agent can encapsulate a SA.

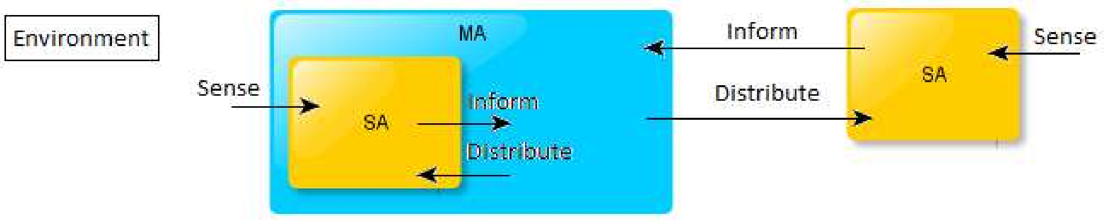


Figure 1-1: Master Agent's structure

The master agents have sufficient capabilities for storing data and CPU. The performance of the whole system depends on the analogy between the master and the simple agents (MAs/SAs), and the average number of simple agents that each master agent supports (average master agent effort).

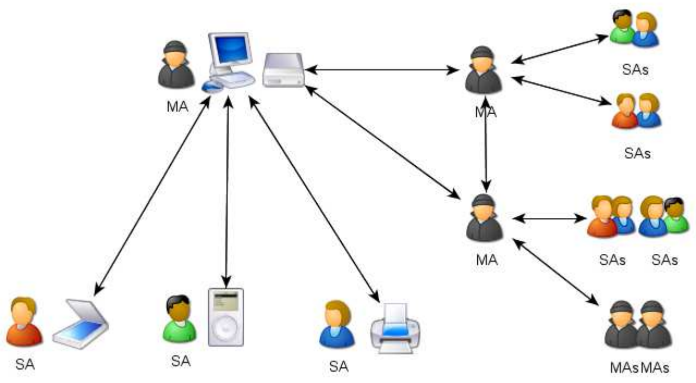


Figure 1-2: Multi-Agent System

In this study, we examine applications for a multi-agent reasoning system where agents are honest, perform absolute cooperation with all other agents and no agent possesses secret goals and aspirations, except from maintaining locally protected knowledge that is not distributed. For example we consider applications for AMI such as Smart Homes or Classrooms. No consideration has been taken for game theory issues between agents. We also make the assumption that the agents share common context representation and data definitions for the pieces of information that they distribute and commonly use. We propose a reasoning process that is able to deal with the demands of a real system and a proof of evidence mechanism which enables us to resolve the conflicts that may occur due to knowledge distribution.

2. Background Theory

2.1 Agents

2.1.1 Intelligent Agents

[45] In artificial intelligence, an **intelligent agent** is an autonomous entity which observes and acts upon an environment and directs its activity towards achieving goals. Intelligent agents may also learn or use knowledge to achieve their goals. They may be very simple or very complex. Intelligent agents are closely related to software agents. In computer science, the term intelligent agent may be used to refer to a software agent that has some intelligence. Yet, intelligent agents are not just software programs, they may also be machines, human beings, communities of human beings, robots or anything that is capable of goal directed behavior.

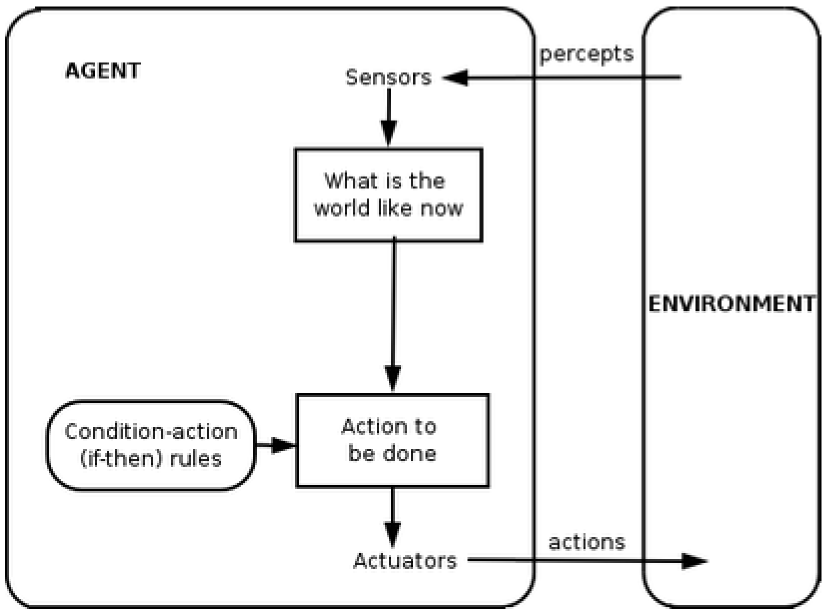


Figure 2-1: Intelligent Agent

There are many definitions for intelligent agents. According to Nikola Kasobov (1998) Intelligent agent systems should exhibit the following characteristics:

- accommodate new problem solving rules incrementally
- adapt online and in real time
- be able to analyze itself in terms of behavior, error and success
- learn and improve through interaction with the environment
- learn quickly from large amounts of data
- have memory-based exemplar storage and retrieval capabilities

- have parameters to represent short and long term memory, age, forgetting, etc.

According to Russell and Norving (2003) agents can group into five classes, based on their degree of perceived intelligence and capability:

1. ***simple reflex agents:*** act only on the basis of the current percept, ignoring the rest of the percept history
2. ***model-based reflex agents:*** can handle a partially observable environment, store the current state inside the agent by maintaining some kind of structure which describes the part of the world which can't be seen
3. ***goal-based agents:*** expand the model-based agents capabilities, by using 'goal' information
4. ***utility-based agents:*** only distinguish between goal states and non-goal states
5. ***learning agents:*** enable agents to initially operate in unknown environments and to become more competent than its initial knowledge alone might allow.

To actively perform their functions, intelligent agents today are normally gathered in a hierarchical structure containing many 'sub-agents'. Intelligent sub-agents process and perform lower level functions. Taken together, the intelligent agent and sub-agents create a complete system that can accomplish difficult tasks of goals with behaviors and responses that display a form of intelligence.

2.1.2 Software Agents

[46] In computer science, a **software agent** is a piece of software that acts for a user or other program in a relationship of agency – the authority to decide which (and if) action is appropriate. The basic concepts with software agents are that they:

- are not strictly invoked for a task but activate themselves
- may reside in wait status on a host, perceiving context
- may get to run status on a host upon starting conditions
- do not require interaction of user
- may invoke other tasks including communication

Here, the term 'agent' describes a software abstraction similar to OOP terms such as methods, functions and objects. The concept of an agent provides a convenient and powerful way to describe a complex software entity that is capable of acting with a certain degree of autonomy in order to accomplish tasks on behalf of its host. But unlike objects, which are defined in terms of methods and attributes, an agent is defined in terms of its behavior. There exist several definitions of software agents which commonly include concepts such as:

- ***persistence:*** code is not executed in demand but runs continuously and decides for itself when is should perform some activity

- ***autonomy:*** agents have capabilities of task selection, prioritization, goal-directed behavior, decision-making without human intervention
- ***social ability:*** agents are able to engage other components through some sort of communication and coordination, they may collaborate on a task
- ***reactivity:*** agents perceive the context in which they operate and react to it appropriately

Franklin and Graesser (1997) discuss four key notions that distinguish agents from arbitrary programs:

1. reaction to the environment
2. autonomy
3. goal-orientation
4. persistence

2.1.3 Multi-Agent Systems (MAS)

[47] A **multi-agent system (MAS)** is a system composed of multiple interacting intelligent agents. Multi-agent systems can be used to solve problems that are difficult or impossible for an individual agent or a monolithic system to solve. Intelligence may include some methodic, functional, procedural or algorithmic search, find and processing approach. Topics where multi-agent systems research may deliver an appropriate approach include online trading, disaster response and modeling social structures.

The agents in a multi-agent system have several important characteristics like:

- ***autonomy:*** the agents are at least partially autonomous
- ***local views:*** no agent has a full global view of the system, or the system is too complex for an agent to make practical use of such knowledge
- ***decentralization:*** there is no designated controlling agent

Typically multi-agent systems research refers to software agent. However, the agents could equally well be robots, humans or human teams. A MAS may contain combined human-agent teams.

When agents can share knowledge using any agreed language, within the constraints of the system's communication protocol, the approach may lead to a common improvement. Example languages are Knowledge Query Manipulation Language (KQML) or FIPA's Agent Communication Language (ACL).

MAS systems, also referred to as 'self-organized systems', tend to find the best solution for their problems 'without intervention'. The main feature which is achieved when developing multi-agent systems is flexibility, since a multi-agent system can be added to, modified and reconstructed, without the need for detailed rewriting of the application. These systems also tend to be rapidly self-recovering and failure proof.

With the development of MAS several methodologies have been proposed for analysis and design agent-oriented systems. In [14] the Gaia methodology combined with Agent UML (AUML) is reported as the more suitable one for agent development and specifically for AMI system development. Gaia methodology is both general and comprehensive, in that it deals with both the macro-level(societal) and the micro-level (agent) aspects of systems. It is founded on the view of a multi-agent system as a computational organization consisting of various interacting roles. In [15], Gaia is illustrated through a case study of an agent-based business process management system. Moraitis and Spanoudakis [16] combine Gaia and JADE for multi-agent system development. Tools like the Organization-based Multiagent System Engineering (O-MaSE) and the agentTool III (aT3) can help their users to analyze, design and implement multi-agent systems.

2.1.4 Agent Communication Language (ACL)

Agent Communication Language (ACL), proposed by the Foundation for Intelligent Physical Agents (FIPA), is a proposed standard language for agent communications. The most popular ACLs are:

- FIPA-ACL
- KQML

Both rely on speech act theory developed by Searle (1960) and enhanced by Winograd and Flores (1970). They define a set of performatives and their meaning. To make agents understand each other they have not only speak the same language, but also have a common ontology. Ontology is a part of the agent's knowledge base that describes what kind of things an agent can deal with and how they are related to each other.

In order to communicate, agents will exchange messages with well-defined structured content.

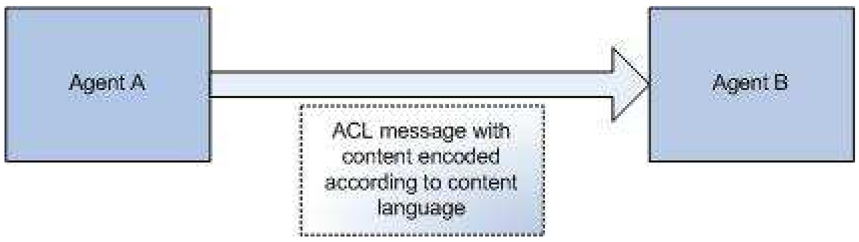

Agent A
Agent B
ACL message with content encoded according to content language

Figure 2-2: Agents' communication through ACL messages

An ACL message has several sections which specify:

- Type of communication act
- Participants of communication
- Content of Message
- Description of Content
- Control of Conversation

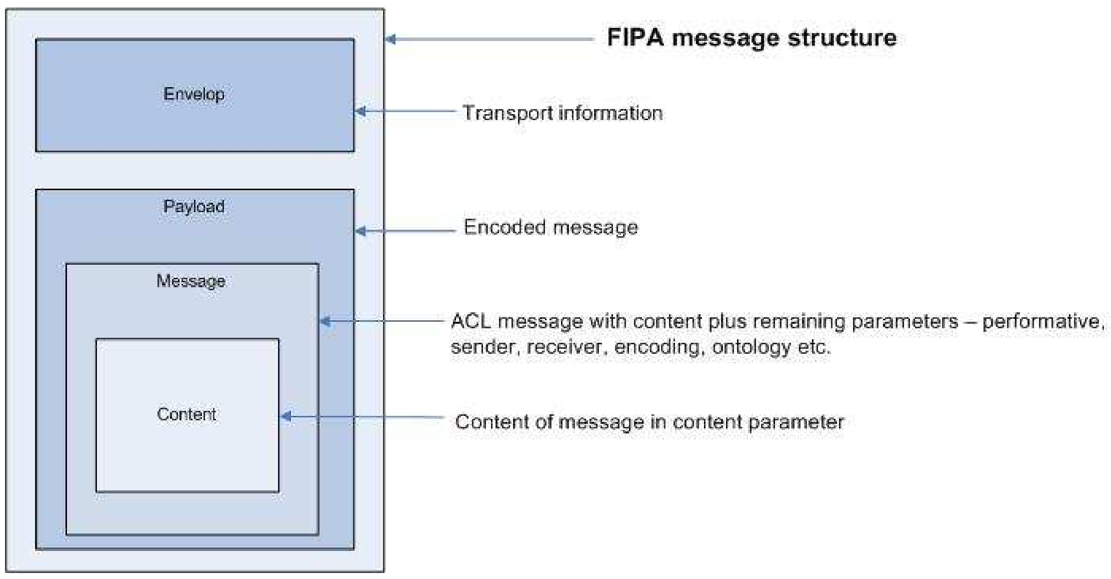

FIPA message structure
Envelop
Transport information
Payload
Encoded message
Message
ACL message with content plus remaining parameters – performative, sender, receiver, encoding, ontology etc.
Content
Content of message in content parameter

Figure 2-3: ACL Message's Structure

In this study, *we will use JADE* which is a framework that implements *FIPA-ACL*.

2.2 Ambient Intelligence (AmI)

2.2.1 Description

[48] In computing, ambient intelligence (AmI) refers to electronic environments that are sensitive and responsive to the presence of people. Ambient intelligence is a vision on the future of consumer electronics, telecommunications and computing that was originally developed in the late 1990s for the time frame 2010-2020. In an ambient intelligence world, devices work in concert to support people in carrying out their everyday life activities, tasks and rituals in easy, natural way using information and intelligence that is hidden in the network connecting these devices. As these devices grow smaller, more connected and more integrated into our

environment, the technology disappears into our surroundings until only the user interface remains perceivable by users.

The ambient intelligence paradigm builds upon pervasive computing, ubiquitous computing, profiling practices, context awareness and human-centric computer interaction design and is characterized by systems and technologies that are:

- ***embedded:*** many networked devices are integrated into the environment
- ***context aware:*** these devices can recognize you and your situational context
- ***personalized:*** they can be tailored to your needs
- ***adaptive:*** they can change in response to you
- ***anticipatory:*** they can anticipate your desires without conscious mediation

Ambient intelligence is closely related to the long term vision of an intelligent service system in which technologies are able to automate a platform embedding the required devices for powering context aware, personalized, adaptive and anticipatory services.

The exploitation is expected to have immediate utility in an abundance of industrial, medical, civil and safety applications. Among the application domain are for instance:

- ***Health systems:*** for better individual health monitoring and tools for health professionals
- ***Safety and security:*** monitoring safety-critical systems and providing digital safeguards
- ***Transport:*** improving safety, efficiency and comfort
- ***E-inclusion:*** supporting people with special needs and including all sectors of society
- ***E-work:*** new methods of work, team work and mobile workers
- ***Socialization:*** nurturing and strengthening social relationships
- ***Sanctuary:*** improve people’s environment to afford more relaxation and personal sanctuary

2.2.2 Criticism

As far as dissemination of information on personal presence is out of control, ambient intelligence vision is subject of criticism. Any immersive, personalized, context-aware and anticipatory characteristics bring up societal, political and cultural concerns about the loss of privacy, as soon as any third party gets control over the respective information and status data.

However, any disabled person may welcome the implicit information presentation and access to improve support and individual assistance. Hence there must be a distinction between solutions for personal improvement and any other purpose.

Power concentration in large organizations, a decreasingly private, fragmented society and hyper real environments where the virtual is indistinguishable from the real are the main topics of critics. Several research groups and communities are investigating the social-economical, political and cultural aspects of ambient intelligence. New thinking on Ambient Intelligence distances itself therefore from some of the original characteristics such as adaptive and anticipatory behavior and emphasizes empowerment and participation to place control in the hands of people instead of organizations.

As long as there is no legal obligation to open one’s individual status data to any access by third party, the degree of freedom still is to stay away of any such solutions and all services with inherited methods of that type.

2.2.3 Social and political aspects

The ISTAG advisory group suggests that the following characteristics will permit the societal acceptance of ambient intelligence:

- AmI should facilitate human contact
- AmI should be oriented towards community and cultural enhancement
- AmI should help to build knowledge and skills for work, better quality of work, citizenship and consumer choice
- AmI should inspire trust and confidence
- AmI should be consistent with long term sustainability – personal, societal and environmental – and with life-long learning
- AmI should be made easy to live with and controllable by ordinary people

2.3 Epistemic Reasoning

Reasoning about the knowledge of an agent and the combined knowledge of a group of agents, referred to in general as *epistemic reasoning*, is a fundamental aspect of reasoning and planning in multi-agent domains. The standard semantics of epistemic reasoning are given by the Kripke structures of modal logic, and are based on the idea of possible worlds. Intuitively, if an agent is unsure about the state of the world, then there must be several candidate states of the world that it considers possible. The agent is then said to know a proposition if it is true in all worlds considered possible. Consider the following scenario:

> “Bob always goes to work on foot, regardless of the weather conditions. Alice knows that if it is raining, Bob will take an umbrella, otherwise he won’t. We consider that the weather will be either sunny or rainy”

If Alice doesn’t have a clue about the weather conditions this morning, she can’t determine if Bob has taken an umbrella. So, Alice maintains two possible worlds: in the first one, it was raining and Bob took an umbrella with him, and in the second one it was sun shining and Bob didn’t take an umbrella. But in both worlds Alice knows that Bob went to work on foot. This proposition is true in all worlds that are considered possible and Alice knows that proposition.

While this works well for a static knowledge base, there is also the question of how each agent's knowledge changes over time as actions are performed in the world. A system is considered to have a set of possible events which could occur at any given instant, and the system state at any time is determined by the sequence of events that have occurred. For each agent there is some subset of these events that it is capable of observing, and it thus has a restricted local view of the state of the system. From the agent's perspective, the system may be in any one of the states that would be compatible with its current local view. If some proposition holds in all such states, then the agent knows that proposition [11][12][13][49].

2.3.1 Epistemic Logic

[52] Epistemic logic is the logic of knowledge and belief. It provides insight into the properties of individual knowers, has provided a means to model complicated scenarios involving groups of knowers and has improved our understanding of the dynamics of inquiry.

Epistemic logic gets its start with the recognition that expressions like ‘knows that’ or ‘believes that’ have systematic properties that are amenable to formal study, In addition to its relevance for traditional philosophical problems, epistemic logic has many applications in computer science and economics. Examples range from robotics, network security and cryptography applications to the study of social and coalitional interactions of various kinds.

For the most part, epistemic logic focuses on propositional knowledge. Here, an agent or a group of agents bears the propositional attitude *knowing* towards some proposition. So, when one says: “Bob knows that there is a chicken in the yard”, one asserts that Bob is the agent who bears the propositional attitude *knowing* towards the proposition expressed by “there is a chicken in the yard”. Beyond straightforward propositional knowledge of this kind, epistemic logic also suggests ways to systematize the logic questions and answers (Bob knows why Murphy barked) and provides insight into the relationships between multiple modes of identification (Bob knows that this man is the chief) and also perhaps event into questions of procedural "know-how". Epistemic logicians have found ways to formally treat a wide variety of knowledge claims in propositional terms.

Syntactically, the language of propositional epistemic logic is simply a matter of augmenting the language of propositional logic with a unary epistemic operator K_c such that

K_cA reads *"Agent c knows A"*

for some arbitrary proposition A.

Hintikka provided a semantic interpretation of epistemic and doxastic operators which we can present in terms of standard possible world semantics along the following lines (Hintikka 1962)

K_cA: in all possible worlds compatible with what c knows, it is the case that A

The basic assumption is that any ascription of propositional attitudes like knowledge, involves dividing the set of possible worlds into two: Those worlds compatible with the attitude in question and those that are incompatible with it.

The set of worlds accessible to an agent depends on his or her informational resources at that instant. It is possible to capture this dependency by introducing a relation of accessibility, R, in the set of possible worlds. To express the idea that for agent c, the world w' is compatible with his information state, or accessible from the possible world w which c is currently in, it is required that R holds between w and w'. This relation is written Rww' and reads "world w' is accessible from w". The world w' is said to be an *epistemic* or *doxastic alternative* to world w for agent c, depending on whether knowledge or belief is the considered attitude. Given the above semantical interpretation, if a proposition A is true in all worlds which agent c considers possible then c knows A.

A possible world semantics for a propositional epistemic logic with a single agent c then consists of a *frame* F which in turn is a pair $\langle W, Rc \rangle$ such that W is a non-empty set of possible worlds and R_c is a binary accessibility relation (relative to agent c) over W. A model $\mathbf{M}$ for an epistemic system consists of a frame and a denotation function φ assigning sets of worlds to atomic propositional formulas. Propositions are taken to be sets of possible worlds; namely the set of possible worlds in which they are true. Let *atom* be the set of atomic propositional formulae, then φ: $atom \mapsto P(W)$, where P denotes the powerset operation. The model $M = \langle W, Rc, \varphi \rangle$ is called a Kripke-model and the resulting semantics Kripke-semantics (Kripke 1963): An atomic formula, **a**, is said to be true in a world w in **M** (written $M, w \vDash \mathbf{a}$) iff w is in the set of possible worlds assigned to **a**, i.e., $M, w \vDash \text{a} \;\; iff \; w \in \varphi(\mathbf{a}) for \; all \; \mathbf{a} \in atom$. The formula K_cA is true in an world w (i.e., $M, w \vDash K_cA$) $iff \; \forall w' \in$

$M, if\ Rww'$ then $M, w' \vDash A$. The semantic for the Boolean connectives follow the usual recursive recipe. A modal formula is said to be *valid* in a frame iff the formula is true for all possible assignments in all worlds in the frame.

Single-agent systems may be extended to groups or multi-agent systems. We can syntactically augment the language of propositional logic with n knowledge operators, one for each agent involved in the group of agents under consideration. The primary difference between the semantics given for a mono-agent and multi-agent semantics is roughly that n accessibility relations are introduced. A modal system for n agents is obtained by joining together n modal logics where for simplicity it may be assumed that the agents are homogenous in the sense that they may all be described by the same logical system. An epistemic logic for *n* agents consists of *n* copies of a certain modal logic. In such an extended epistemic logic it is possible to express that some agent in the group knows a certain fact, that an agent knows that another agent knows a fact etc. It is possible to develop the logic even further. Not only may an agent know that another agent knows a fact, but they may all know this fact simultaneously. From here it is possible to express that everyone knows that everyone knows that everyone knows, that … That it is *common knowledge*.

As Lewis noted in his book *Convention* (1969) a convention requires common knowledge among the agents that observe it. A variety of norms, social and linguistic practices, agent interactions and game presuppose common knowledge (Aumann 1994). A relatively simple way of defining common knowledge is not to partition the group of agents into subsets with different common 'knowledges' but only to define common knowledge for the entire group of agents. Once multiple agent have been added to syntax, the language is augmented with an additional operator *c*. *CA* is then interpreted as 'It is common knowledge among the agents that *A*'. Well-formed formulas follow the standard recursive recipe with a few, but obvious, modifications taking into account the multiple agents. An auxiliary operator *E* is also introduced such that *EA* means 'Everyone knows that *A*'. *EA* is defined as the conjunction $K_1A \wedge K_2A \wedge \ldots \wedge K_nA$.

To semantically interpret *n* knowledge operators, binary accessibility relations R_n are defined over the set of possible worlds *W*. A special accessibility relation, R^o, is introduced to interpret the operator of common knowledge. The relation must be flexible enough to express the relationship between individual and common knowledge. The idea is to let the accessibility relation for c be the transitive closure of the union of the accessibility relations corresponding to the singular knowledge operators.

We indicate below the formal definitions for epistemic languages, models and semantics (adopted by [12],[11]). We use a countable set of proposition letters P and a finite set of agents A.

Languages

The basic epistemic language consists of all formulas that can be built from proposition letters in P using conjunction, negation and a modal operator K_a for every agent aEA. $K_a\varphi$ stands for *agent a knows that φ is true*. The basic epistemic language is denoted by L_K:

$$\Phi \ := \ p|\ \varphi \wedge \psi \mid \neg\varphi \mid K_\alpha\varphi \ |D_B\varphi$$

where $p \in P, a \in A, and\ B \subseteq A.$

Models

A model M is a triple (W, R, V), where:

W is a non-empty set of worlds

$R: A \to \wp(W \times W)$

$V: W \to \wp(P)$

R assigns to every agent $a \in A$ a so-called accessibility relation on W.

Semantics

The satisfaction relation |= between pointed models and formulas in L_K or L_D is recursively defined as follows:

$M, w \vDash p$	$iff\ p \in V(w)$
$M, w \vDash \neg\varphi$	$iff\ M, w\ \nvDash \varphi$
$M, w \vDash \varphi \wedge \psi$	$iff\ M, w \vDash \varphi\ and\ M, w \vDash \psi$
$M, w \vDash K_\alpha\varphi$	$iff\ M, u \vDash \varphi\ for\ all\ \ u \in [M, w]_\alpha$
$M, w \vDash D_B\varphi$	$iff\ M, u \vDash \varphi\ for\ all\ \ u \in [M, w]_B$

The $K_a\varphi$ clause says that an agent knows φ to be true just in case φ is true in all worlds he considers possible.

The $D_B\varphi$ clause says that φ is distributed knowledge among B just in case φ is true in all worlds that every agent in B considers possible.

2.4 Distributed Knowledge

A formula φ is distributed knowledge among a group of agents B iff φ follows from the knowledge of all individual agents in B put together. Semantically, φ is distributed knowledge among B iff φ is true in all worlds that every agent in B considers possible.

The notion of distributed knowledge is used to express what a group of agents would know if they were to combine their information. An article that is read by the readers of a newspaper is distributed knowledge.

2.4.1 Full communication

Whenever φ is considered *group knowledge*, it should be possible for the members of the group to *establish φ through communication*. The newspaper article can't consider as group knowledge among *all readers* of the article as they cannot establish that knowledge through communication. However, the members of a smaller fellowship that read the article and make comments about it, consider the article as group knowledge.

2.4.2 Commonsense Reasoning

Commonsense reasoning is a process that involves taking information about certain aspects of a scenario in the world and making inferences about other aspects of the scenario based on our *commonsense knowledge*, or knowledge of how the world works. If we want to study an animal that we know that is a fish, we can infer that it lives in the water. Commonsense reasoning is essential to intelligent behavior and thought. It allows us to fill in the blanks, to reconstruct missing portions of a scenario, to figure out what happened, and to predict what might happen next.

When we perform commonsense reasoning, we rarely have complete information. We are unlikely to know the state of affairs down to the last detail, everything about the events that are occurring, or everything about the way the world works. Therefore, when we perform commonsense reasoning, we must jump to conclusions. Yet, if new information becomes available that invalidates those conclusions, then we must also be able to take them back. Reasoning in which we reach conclusions and retract those conclusions when warranted is known as *default reasoning*. We know that by default birds can fly. An animal that is classified as bird should have the ability to fly. If we are said that Memphis is a bird, we conclude that it can fly unless we have evidence for the opposite. If we are later informed that

Memphis is a penguin, we will retract our conclusion about its flying capability, as penguins cannot fly. If we were known from the first time that Memphis was a penguin, we would infer that is doesn't fly based on our commonsense knowledge.

2.4.3 Default reasoning

Default reasoning is reasoning in which we reach a conclusion based on limited information, and possibly later retracts that conclusion when new information comes in. The event calculus supports default reasoning using circumscription, which can be used to minimize unexpected events, minimize unexpected effects of events, and minimize exceptional conditions.

2.4.4 The closed world assumption

[49] The *closed world assumption* (CWA) is the presumption that what is not currently known to be true, is false. The same name also refers to a logical formalization of this assumption by Raymond Reiter. The opposite of the closed world assumption is the open world assumption (OWA), stating that lack of knowledge does not imply falsity. Decisions on CWA vs.OWA determine the understanding of the actual semantics of a conceptual expression with the same notations of concepts. A successful formalization of natural language semantics usually can not avoid an explicit revelation of the implicit logical backgrounds based on whether CWA or OWA.

Negation as failure (NAF) is related to the closed world assumption, as it amounts to believing false every predicate that cannot be proved to be true. NAF is a non-monotonic inference rule in logic programming, used to derive $not\ p$ (i.e. that p is assumed not to hold) from failure to derive p. Note that not p can be different from the statement $\neg p$ of the logical negation of p, depending on the completeness of the inference algorithm and thus also on the formal logic system. For example, negation as failure could be implemented as follows:

$$if\ (not\ (goal\ p)), then\ (assert\ \neg p)$$

Which says that if the goal to prove p fails, then assert $\neg p$.

In the knowledge management arena, the closed world assumption is used in at least two situations:

1. When the knowledge base is known to be complete (e.g., a corporate database containing records for every employee), and
2. When the knowledge base is known to be incomplete but a "best" definite answer must be derived from incomplete information.

For example, if a database contains the following table reporting editors who have worked on a given article, a query on the people not having edited the article on Formal Logic is usually expected to return “Sarah Johnson”.

Edit	
Editor	**Article**
John Doe	Formal logic
John Doe	Closed World Assumption
Joshua A. Norton	Formal Logic
Sarah Johnson	Introduction to Spatial Databases
Charles Ponzi	Formal Logic
Emma Lee-Choon	Formal Logic

Table 2-4: Example table ‘Edit’

In the closed world assumption, the table is assumed to be complete (it lists all editor-article relationships), and Sarah Johnson is the only editor who has not edited the article on Formal Logic. In contrast, with the open world assumption the table is not assumed to contain all editor-article tuples, and the answer to who has not edited the Formal Logic article is unknown. There is an unknown number of editors not listed in the table, and an unknown number of articles edited by Sarah Johnson that are also not listed in the table.

2.4.5 Circumscription

[51] *Circumscription* is a non-monotonic logic created by John McCarthy to formalize the common sense assumption that things are as expected unless otherwise specified. Circumscription was later used by McCarthy in an attempt to solve the frame problem. In its original first-order logic formulation, circumscription minimizes the extension of some predicates, where the extension of a predicate is the set of tuples of values the predicate is true on. This minimization is similar to the closed world assumption that what is not known to be true is false.

The original problem considered by McCarthy was that of missionaries and cannibals:

> “There are three missionaries and three cannibals on one bank of a river; they have to cross the river using a boat that can only take two, with the additional constraint that cannibals must never outnumber the missionaries on either bank (as otherwise the missionaries would be killed and, presumably, eaten).”

The problem considered by McCarthy was not that of finding a sequence of steps to reach the goal (the article on the missionaries and cannibals problem contains one such solution), but rather that of excluding conditions that are not explicitly stated.

For example, the solution “go half a mile south and cross the river on the bridge” is intuitively not valid because the statement of the problem does not mention such a bridge. On the other hand, the existence of this bridge is not excluded by the statement of the problem either. That the bridge does not exist is a consequence of the implicit assumption that the statement of the problem contains everything that is relevant to its solution. Explicitly stating that a bridge does not exist is not a solution to this problem, as there are many other exceptional conditions that should be excluded (such as the presence of a rope for fastening the cannibals, the presence of a larger boat nearby, etc.)

Circumscription was later used by McCarthy to formalize the implicit assumption of inertia: things do not change unless otherwise specified. Circumscription seemed to be useful to avoid specifying that conditions are not changed by all actions except those explicitly known to change them; this is known as the frame problem. However, the solution proposed by McCarthy was later shown leading to wrong results in some cases, like in the Yale shooting problem scenario. Other solutions to the frame problem that correctly formalize the Yale shooting problem exist; some use circumscription but in a different way.

2.4.6 Commonsense law of inertia

A quality of the commonsense world is that objects tend to stay in the same state unless they are affected by events. An event typically changes only a small number of things and everything else in the world remains unchanged. A book sitting on a table remains on the table unless it is picked up, a light stays on until it is turned off, and a falling object continues to fall until it hits something. This is known as the commonsense law of inertia.

2.4.7 The Frame Problem

The problem of representing and reasoning about which fluents do not change when an event occurs is known as the *frame problem.* Consider a room with one light and one door. When we turn on the light what we can say about the door’s state?

The frame problem is that specifying only which conditions are changed by the actions, do not allow, in logic, to conclude that all other conditions are not changed. This problem could be solved by adding frame axioms, which explicitly specify that all conditions not affected by actions are not changed while executing that action. For example, since the action executed at time 0 is that of opening the door, a frame axiom would state that the status of the light does not change from time 0 to time 1. The frame problem is that one such frame axiom is necessary for every pair of action and condition such that the action does not affect the condition. The number of these axioms in the general case is$2 \times A \times F$, where A is the number of actions and F is the number of fluent. In other words, the problem is that of formalizing a dynamical domain without explicitly specifying the frame axioms.

The solution within Event Calculus is the use of *inertia* and *circumscription*. In the Event Calculus, fluents may either be subject to inertia or released from it, according to context. In the EC, inertia is enforced by formulae stating that a fluent is true if it has been true at a given previous time point and no action changing it to false has been performed in the meantime. Predicate completion is still needed in the EC for obtaining that a fluent is made true only if an action making it true has been performed, but also for obtaining that an action had been performed only if that is explicitly stated. The circumscription in the EC assumes *by default* that no unexpected events occur and there are no unexpected effects of events.

2.4.8 Kripke models

Kripke semantics is a formal semantics for non-classical logic systems created by Saul Kripke. The language of propositional modal logic consists of a countable infinite set of propositional variables, a set of truth-factional connectives and the modal operator.

Traditionally, a statement is regarded as logically valid if it is an instance of a logically valid form, where a form is regarded as logically valid if every instance is true. In modern logic, forms are represented by formulas involving letters and special symbols, and logicians seek therefore to define a notion of model and a notion of a formula's truth in a model in such a way that every instance of a form will be true if and only if a formula representing that form is true in every model.

A Kripke frame or modal frame is a pair $\langle W, R \rangle$, where W is a non-empty set, and s is a binary relation on W. Elements of W are called nodes or worlds, and R is known as the accessibility relation. Depending on the properties of the accessibility relation, the corresponding frame is described, by extension, as being transitive, reflexive, etc. A Kripke model is a triple $\langle W, R, \vDash \rangle$, where $\langle b, R \rangle$ is a Kripke frame, and $\vDash$ is a relation between nodes of W and modal formulas. A truth of a modal sentence is evaluated relative to a particular possible world w in a particular Kripke structure$\langle W, R \rangle$. The satisfaction relation is defined recursively as follows:

$M, w \vDash p$ $\quad$ $if\ p\ is\ true\ in\ w, for\ any\ primitive\ proposition$

$M, w \vDash \varphi \wedge \psi$ $\quad$ $iff\ M, w \vDash \varphi\ and\ M, w \vDash \psi$

$M, w \vDash \neg\varphi$ $\quad$ $iff\ it\ is\ not\ the\ case\ that\ M, w\ \vDash \varphi$

$$M,w \vDash \blacksquare\varphi \qquad iff\ for\ any\ w' \in W\ such\ that\ R(w,w'),$$
$$it\ is\ the\ case\ that\ \ M,w' \vDash \varphi$$

Axiom 2.4.8.1 (Classical) All propositional tautologies are valid

Axiom 2.4.8.2 (K) $\left(\blacksquare\varphi \wedge \blacksquare(\varphi \rightarrow \psi)\right) \rightarrow \blacksquare\psi$ is valid

Rule 2.4.8.3 (Modus Ponens) if both φ and $\varphi \rightarrow \psi$ are valid, infer the validity of ψ

Rule 2.4.8.4 (Necessitation) From the validity of φ infer the validity of $\blacksquare\varphi$.

Theorem 2.4.8.5 The system K is sound and complete for the class of all Kripke models.

2.4.9 Definitions

(The definitions and the axioms are adopted by [13])

Definition 2.4.9.1 ("Everyone knows") Let M be a Kripke structure, w be a possible world in M, G be a group of agents, and φ be a sentence of modal logic. Then $M,w \vDash E_G\varphi$ if and only if $M,w' \vDash \varphi$ for all $w' \in \cup_{i\in G} I_i(w)$. (Equivalently, we can require that $\forall i \in G$ it is the case that $M,w \vDash K_i\varphi$)

In other words, everybody knows a sentence when the sentence is true in all of the worlds that are considered possible in the current world by any agent in the group. A lecture that was given to three students A,B and C is a piece of knowledge that everyone of these students knows.

Definition 2.4.9.2 (Common knowledge) Let M be a Kripke structure, w be a possible world in M, G be a group of agents, and φ be a sentence of model logic. Then $M,w \vDash C_G\varphi$ if and only if $M,w \vDash E_G(\varphi \wedge C_G\varphi)$

In other words, a sentence is common knowledge if everybody knows it and knows that it is common knowledge. In the example above, consider that student C informs student D, which was absent from the classroom, about the content of the lecture. The content of the lecture is a piece of knowledge that everyone knows but it is not common knowledge among all the students, as students A and B ignore the fact that D was informed. The content of the lecture is common knowledge among A,B and C and among C and D separately. In order the lecture's content to become common knowledge among all students, the following events must take place:

1. A and B must be informed that D knows the lecture's content and so everyone in the group knows that
2. A, B, C and D must clarify through communication that the lecture's content is common knowledge among them

So, every participant:

- Would know that statement
- Would know that all the other participants know that statement too
- Would know that all of them knows all the above
- Would know that all the participants consider the statement as common knowledge

Fixed-point axiom: A sentence is common knowledge if everybody knows it and knows that is common knowledge. This formula is called the *fixed-point axiom,* since $C_G\varphi$ can be viewed as the fixed point solution of the equation $f(x) = E_G\big(\varphi \wedge f(x)\big)$. Fixed-point definitions are notoriously hard to understand intuitively. Fortunately, we can give alternative characterizations of C_G. First, we can give a direct semantic definition.

Theorem 2.4.9.3 Let M be a Kripke structure, w be a possible world in M, G be a group of agents, and φ be a sentence of modal logic. Then $M, w \vDash C_G\varphi$ if and only if $M, w' \vDash \varphi$ for every sequence of possible worlds ($w = w_0$, w_1,…, $w_n = w'$) for which the following holds: for every $0 \leq i \leq n$ there exists an agent $j \in G$ such that $w_{i+1} \in I_j(w_i)$.

Second it is worth noting that a S5 axiomatic system can also enriched to provide a sound and complete axiomatization of C_G. It turns out that what are needed are two additional axioms and one new inference rule.

Axiom 13.4.4 (A3) $E_G\varphi \leftrightarrow \bigwedge_{i\in G} K_i\varphi$

Axiom 13.4.5 (A4) $C_G\varphi \rightarrow E_G(\varphi \wedge C_G\varphi)$

Rule 13.4.6 (R3) $From\ \varphi \rightarrow E_G(\psi \wedge \varphi)\ infer\ \varphi \rightarrow C_G\psi$

This last inference rule is a form of an *induction rule.*

2.4.10 Our approach

In our system we achieve ***full communication*** among agents through the agent's network capabilities. We ***distribute knowledge*** by publishing knowledge to MAs. Since every piece of information that is published, is globally accessible by every agent (***everybody knows***) and every agent knows that all the other agents know this piece of information too, we achieve ***common knowledge*** for every piece of knowledge that is published. An agent can maintain locally protected knowledge that is not distributed to the system. We suppose that these pieces of knowledge are originated from local information that is processed by the specific agent and is not relevant with common knowledge.

We use Event Calculus to perform ***default reasoning*** and ***achieve commonsense reasoning***. We use DECKT (Discrete Event Calculus Knowledge Theory) to perform ***epistemic reasoning*** for Event Calculus.

2.5 DEC (Discrete Event Calculus)

2.5.1 Description

The **event calculus (EC)** is formalism for reasoning about action and change. The event calculus has actions, which are called *events* and indicate changes in the environment, time-varying properties or *fluent* and a timepoint sort that implements a linear time structure on which actual events occur. It is based on first-order predicate calculus and is capable for simulating a variety of phenomena such as actions with indirect effects, actions with non-deterministic effects, compound actions, concurrent actions and continuous change. The EC defines predicates for expressing, among others, which fluents hold when (*HoldsAt*), what events happen (*Happens*) and which their effects are (*Initiates*, *Terminates*). It adopts a straightforward solution to the frame problem which is robust and works in the presence of each of these phenomena.

The event calculus supports context-sensitive effects of events, indirect effects, action preconditions, and the commonsense law of inertia. Certain phenomena are addressed more naturally in the event calculus, including concurrent events, continuous time, continuous change, events with duration, nondeterministic effects, partially ordered events, and triggered events. Examples for such phenomena could be:

- The *commonsense law of inertia* – when moving a glass does not cause a glass in another room to move.
- *Conditional effects of events* – the results of turning on a television set depend on whether or not it is plugged in
- *Release from the commonsense law of inertia* – if a person is holding a PDA, then the location of the PDA is released from the commonsense law of inertia so that the location of the PDA is permitted to vary
- *Event ramifications or indirect effects of events* – the PDA moves along with the person holding it (state constraint) or instantaneous propagation of

interacting indirect effects, as in idealized electrical circuits (casual constraints)

- *Events with nondeterministic effects* – when flipping a coin results in the coin landing either heads or tails
- *Gradual change* – the changing height of a falling object or volume of a balloon in the process of inflation
- *Triggered events or events that are triggered under certain conditions* – if water is flowing from a faucet into a sink, then once the water reaches a certain level the water will overflow
- *Concurrent events with cumulative or cancelling effects* – if a shopping cart is simultaneously pulled and pushed then it will spin around

In order to model a *domain description* we need to define:

- An *axiomatization* describing a commonsense domain or other domains of interest
- *Observations* of world properties at various times
- A *narrative* of known event occurrences

We use a simple example to illustrate what the event calculus does. Suppose we wish to reason about turning on and off a light. We start by representing general knowledge about the effects of events:

If a light's switch is flipped up, then the light will be on.
If a light's switch is flipped down, then the light will be off.

We then represent a specific scenario:

The light was off at time 0.
Then the light's switch was flipped up at time 2.
Then the light's switch was flipped down at time 4.

We use the event calculus to conclude the following:

At time 1, the light was off.
At time 3, the light was on.
At time 5, the light was off.

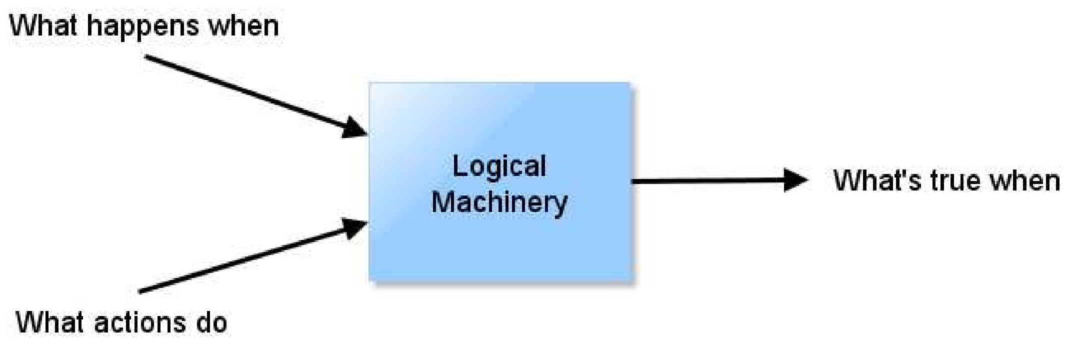


Figure 2-5: The deduction task of the Event Calculus

EC is a logical mechanism that infers what's true when given what happens when and what actions do. The 'what happens when' part is a narrative of events and the 'what actions do' part describes the effects of actions. The EC supports several reasoning tasks that can be categorized into *deductive* tasks, *abductive* tasks and *inductive* tasks. For this study we use DECKT which is a Jess implementation of DEC that supports **deduction**. In deduction, an initial state and a sequence of events are given, and the resulting state is tried to be determined. Deduction includes temporal *projection* or *prediction*, where the outcome of a known sequence of actions is sought.

Miller and Shanahan introduced several alternative formulations of the basic event calculus. A number of their axioms can be combined to produce what we call EC, which differs from the basic event calculus. *ReleasedAt*(f, t) represents that fluent f is released from the commonsense law of inertia at timepoint t. *AntiTrajectory*(f_1, t_1, f_2, t_2) represents that, if fluent f_1 is terminated by an event that occurs at timepoint t_1, then fluent f_2 will be true at timepoint $t_1 + t_2$.

Predicate	Meaning
HoldsAt(f,t)	**f** is true at **t**
Happens (e,t)	**e** occurs at **t**
ReleasedAt(f,t)	**f** is released from the commonsense law of inertia at **t**
Initiates(e,f,t)	If **e** occurs at **t**, then **f** is **true** and not released from the commonsense law of inertia after **t**
Terminates(e,f,t)	If **e** occurs at **t**, then **f** is **false** and not released from the commonsense law of inertia after **t**
Releases(e,f,t)	If **e** occurs at **t**, then **f** is **released** from the commonsense law of inertia after **t**
$\mathbf{Trajectory(f_1, t_1, f_2, t_2)}$	If $\mathbf{f_1}$ is initiated by an event that occurs at $\mathbf{t_1}$, then $\mathbf{f_2}$ is **true** at $\mathbf{t_1 + t_2}$
$\mathbf{AntiTrajectory(f_1, t_1, f_2, t_2)}$	If $\mathbf{f_1}$ is terminated by an event that occurs at $\mathbf{t_1}$, then $\mathbf{f_2}$ is **true** at $\mathbf{t_1 + t_2}$

Table 2-6: The predicates of the Event Calculus

The **discrete event calculus (DEC)** improves efficiency of automated reasoning by limiting time to the integers, and eliminating triply quantified time from many of the axioms. The predicates of DEC are the same as those of EC.

2.5.2 DEC Axiomatization

(The axiomatization is adopted by [4]) **DEC** restricts the timepoint sort to the integers. It consists of 12 axioms and definitions.

Stopped and Started

DEFINITION DEC1: $StoppedIn(t_1, f, t_2) \overset{def}{\equiv} \exists e, t\ (Happens(e, t) \wedge t_1 < t < t_2 \wedge\ Terminates(e, f, t))$

DEFINITION DEC2: $StartedIn(t_1, f, t_2) \overset{def}{\equiv} \exists e, t\ (Happens(e, t) \wedge t_1 < t < t_2 \wedge\ Initiates(e, f, t))$

Trajectory and AntiTrajectory

AXIOM DEC3:

$$\big(Happens(e, t_1) \wedge Initiates(e, f_1, t_1) \wedge 0 < t_2 \wedge Trajectory(f_1, t_1, f_2, t_2) \wedge \neg StoppedIn(t_1, f_1, t_1{+}t_2)\big) \Rightarrow HoldsAt(f_2, t_1{+}t_2)$$

AXIOM DEC4:

$$\big(Happens(e, t_1) \wedge Terminates(e, f_1, t_1) \wedge 0 < t_2 \wedge AntiTrajectory(f_1, t_1, f_2, t_2) \wedge \neg StartedIn(t_1, f_1, t_1{+}t_2)\big) \Rightarrow HoldsAt(f_2, t_1{+}t_2)$$

Inertia of HoldsAt

The commonsense law of inertia is enforced for HoldAt.

AXIOM DEC5: $(HoldsAt(f, t) \wedge \neg ReleasedAt(f, t + 1) \wedge \neg\exists e(Happens(e, t) \wedge Terminates(e, f, t))) \Rightarrow HoldsAt(f, t + 1)$

If a fluent is true at timepoint t, the fluent is not released from the commonsense law of inertia at t+1 and the fluent is not terminated by any event that occurs at t, then the fluent is true at t+1.

AXIOM DEC6:

$$(\neg HoldsAt(f, t) \wedge \neg ReleasedAt(f, t + 1) \wedge \neg\exists e(Happens(e, t) \wedge Initiates(e, f, t))) \Rightarrow \neg HoldsAt(f, t + 1)$$

If a fluent is false at timepoint t, the fluent is not released from the commonsense law of inertia at t+1 and the fluent is not initiated by any event that occurs at t, then the fluent is false at t+1.

Inertia of ReleasedAt

Inertia is enforced for ReleasedAt.

AXIOM DEC7: $(ReleasedAt(f, t) \wedge \neg\exists e(Happens(e, t) \wedge (Initiates(e, f, t) \vee Terminates(e, f, t)))) \Rightarrow ReleasedAt(f, t + 1)$

If a fluent is released from the commonsense law of inertia at timepoint t and the fluent is neither initiated nor terminated by any event that occurs at t, then the fluent is released from the commonsense law if inertia at t+1.

AXIOM DEC8:

$$(\neg ReleasedAt(f,t) \land \neg\exists e(Happens(e,t) \land Releases(e,f,t))) \Rightarrow \neg ReleasedAt(f,t+1)$$

If a fluent is not released from the commonsense law of inertia at timepoint t and the fluent is not released by any event that occurs at t, then the fluent is not released from the commonsense law if inertia at t+1.

Influence of Events on Fluents

Event occurrences influence the states of fluents.

AXIOM DEC9: $\big(Happens(e,t) \land Initiates(e,f,t)\big) \Rightarrow HoldsAt(f,t+1)$

If a fluent is initiated by some event that occurs at timepoint t, then the fluent is true at t+1.

AXIOM DEC10: $\big(Happens(e,t) \land Terminates(e,f,t)\big) \Rightarrow \neg HoldsAt(f,t+1)$

If a fluent is terminated by some event that occurs at timepoint t, then the fluent is false at t+1.

AXIOM DEC11: $\big(Happens(e,t) \land Releases(e,f,t)\big) \Rightarrow ReleasedAt(f,t+1)$

If a fluent is released by some event that occurs at timepoint t, then the fluent is released from the commonsense law of inertia at t+1.

AXIOM DEC12: $(Happens(e,t) \land (Initiates(e,f,t) \lor Terminates(e,f,t))) \Rightarrow \neg ReleasedAt(f,t+1)$

If a fluent is initiated or terminated by some event that occurs at timepoint t, then the fluent is not released from the commonsense law of inertia at t+1.

Let DEC be the formula generated by conjoining axioms DEC3 through DEC12 and then expanding the predicates *StoppedIn* and *StartedIn* using definitions DEC1 and DEC2.

2.5.3 Inconsistency in Event Calculus

There are three cases where inconsistency can come up when we:

1. Simultaneously initiating and terminating a fluent
2. Simultaneously releasing and initiating or terminating a fluent
3. Use effect axioms and state constraints

2.6 DECKT (Discrete Event Calculus Knowledge Theory)

[9] The Discrete Event Calculus Knowledge Theory (DECKT), thoroughly described in (Patkos and Plexousakis, 2009), is a formal theory for reasoning about knowledge, actions and causality. It builds on the Event Calculus (Kowalski and Sergot, 1986) and extends it with epistemic capabilities enabling reasoning about a wide range of commonsense phenomena, such as temporal and delayed knowledge effects, knowledge ramifications, concurrency, non- determinism and others.

The theory employs the discrete time Event Calculus axiomatization described in (Mueller,2006). It assumes agents acting in dynamic environments, having accurate but potentially incomplete knowledge and able to perform knowledge-producing actions and actions with context-dependent effects .Knowledge is treated as a fluent, namely the *Knows* fluent, which expresses knowledge about fluent and fluent formulae, obtained either from direct effects of actions or indirectly through ramifications modeled as state constraints .For technical reasons, for direct effects the auxiliary *KP* fluent(for ”knows persistently”) is used that is related with the *Knows* fluent by the axiom1:

(KT2) $\boldsymbol{HoldsAt(KP(f),t) \Rightarrow HoldsAt(Knows(f),t)}$

DECKT augments a domain axiomatization with a set of meta-axioms describing epistemic derivations. For instance, for positive effect axioms that specify under what conditions action *e* initiates fluent f , i.e., $\wedge^{i} HoldsAt(f_i,t) \Rightarrow Initiates(e,f,t)$, DECKT introduces the(KT3) set of axioms expressing that if the conjunction of preconditions $C = \{\overrightarrow{f_i}\}$ is known then after *e* the effect will be known to be true, such as:

(KT3.1)
$\boldsymbol{\wedge^{f_i \in C} HoldsAt(Knows(f_i),t) \wedge Happens(e,t) \Rightarrow Initiates(e,KP(f),t)}$

In a similar fashion, the (KT5) axiom set captures the fact that if some precondition is unknown while none is known to be false, then after *e* knowledge about the effect is lost. The approach proceeds analogously for negative effect axioms $\wedge^{i} HoldsAt(f_i,t) \Rightarrow Terminates(e,f,t)$ and release axioms $\wedge^{i} HoldsAt(f_i,t) \Rightarrow$

$Releases(e, f, t)$. The latter model non-deterministic effects, therefore they result in loss of knowledge about the effect.

Knowledge-producing (sense) actions provide information about the truth value of fluents and, by definition, cause no effect to the environment, instead only affect the mental state of the agent:

(KT4) $\boldsymbol{Initiates(sense(f), KPw(f), t)}$

Kw is an abbreviation for whether a fluent is known (similarly for KPw):

$HoldsAt(Kw(f), t) \equiv HoldsAt(Knows(f), t) \vee HoldsAt(Knows(\neg f), t)$

Furthermore, the theory also axiomatizes so called *hidden causal dependencies* (HCDs). HCDs are created when executing actions with unknown preconditions and capture the fact that in certain cases, although knowledge about the effect is lost, it becomes contingent on the preconditions; obtaining knowledge about the latter through sensing can provide information about whether the effect has actually occurred. Consider the positive effect axiom $HoldsAt(f', t) \Rightarrow Initiates(e, f, t)$, where fluent f' is unknown to the agent and f known to be false at T (f may denote that a door is open, f' that a robot stands in front of that door and e the action of pushing forward gently).If e happens at T, f becomes unknown at $T+1$,asdictatedby(KT5), still a dependency between f' and f must be created to declare that if we later sense any of them we can infer information about the other, as long as no event alters them in the meantime(either the robot was standing in front of the door and opened it or the door remained closed).

DECKT introduces the (KT6) set of axioms that specify when HCDs are created or destroyed and what knowledge is preserved when an HCD is destroyed. For instance, for positive effect axioms (KT6.1.1) below, creates an appropriate implication relation:

(KT6.1.1) $\boldsymbol{\neg HoldsAt(Knows(f), t) \wedge \neg HoldsAt(Knows(\bigvee^{f_i \in C} \neg fi), t) \wedge \bigvee^{f_i \in C}[\neg HoldsAt(Kw(f_i), t)] \Rightarrow Initiates(e, KP(f \vee \bigvee^{f_j \in C(t)^-} \neg f_j), t)}$

where $C(t)^-$ denotes those precondition fluent that are unknown to the agent at timepoint *t*. In a nutshell, DECKT augments the mental state of an agent with a disjunctive knowledge formula, equivalent to $HoldsAt(Knows(\bigwedge^{f_j \in C(t)^-} f_j \Rightarrow f), t+1)$, that encodes a notion of epistemic causality in the sense that if future knowledge brings about $(\bigwedge^{f_j \in C(t)^-} f_j)$ it also brings about f.

DECKT adopts an alternative representation for knowledge that does not employ the possible worlds semantics, which are computationally intensive and not appropriate for practical implementations. Instead, its efficiency stems from the fact that HCDs, which are based on a provably sound and complete translation of possible worlds into implication rules, are treated as ordinary fluents.

2.7 Consistent theory

[53] In logic, a *consistent* theory is one that does not contain a contradiction. The lack of contradiction can be defined in either semantic or syntactic terms. The semantic definition states that a theory is consistent if it has a model, this is the sense used in traditional Aristotelian logic, although in contemporary mathematical logic the term *satisfiable* is used instead. The syntactic definition states that a theory is consistent if there is no formula *P* such that both *P* and its negation are provable from the axioms of the theory under its associated deductive system. If these semantic and syntactic definitions are equivalent for a particular logic, the logic is *complete*. A set of beliefs is consistent just if it would be possible for them to be true together: that is, if they are either in fact all true or could all have been true. A set of beliefs is inconsistent just if it would be impossible for them all to be true.

When our inner systems (beliefs, attitudes, values, etc.) all support one another and when these are also supported by external evidence, then we have a comfortable state of affairs. The discomfort of cognitive dissonance occurs when things fall out of alignment, which leads us to try to achieve a maximum practical level of consistency in our world.

We also have a very strong need to believe we are being consistent with social norms. When there is conflict between behaviors that are consistent with inner systems and behaviors that are consistent with social norms, the potential threat of social exclusion often sways us towards the latter, even though it may cause significant inner dissonance.

2.8 Coherence theory

[54] As a theory of truth, coherentism restricts *true* sentences to those that cohere with some specified set of sentences. Someone's belief is true if and only if it is *coherent* with all or most of his or her other beliefs. Usually, coherence is taken to imply something stronger than mere consistency. Statements that are comprehensive and meet the requirements of Occam's razor are usually to be preferred.

As an illustration of the principle, if people lived in a virtual reality universe, they could see birds in the trees that aren't really there. Not only are the birds not really there, but the trees aren't really there either. The people *know* that the bird and the tree are there, because it coheres with the rest of their experiences in the virtual reality. Talking about *coherence* is an abstract way of talking about the things that the people really know, without regard for whether they are in a virtual reality or not.

Perhaps the best-known objection to a coherence theory of truth is Bertrand Russell's. Russell maintained that since both a belief and its negation will, individually, cohere with at least one set of beliefs, this means that contradictory beliefs can be shown to be true according to coherence theory, and therefore that the theory cannot work. However, what most coherence theorists are concerned with is not all possible beliefs, but the set of beliefs that people actually hold. The main problem for a coherence theory of truth, then, is how to specify just this particular set, given that the truth of which beliefs are actually held can only be determined by means of coherence.

It can be seen that the idea that beliefs correspond to states of affairs is problematic. This is because what we get are not 'states of affairs' at all, but only other perceptions that in turn require foundations. There are 3 main problems with this view:

1. If false beliefs outweigh true ones, this would make the incorrect conclusion the correct one - according to the coherence theory of truth. For instance, if I believe that the 1969 moon landings were faked in a photographic studio, I might be able to back this up with selective evidence. If I then reach a point where the evidence for this is more than for the belief that the moon landings took place, I would be forced to conclude that it was the truth.
2. Coherence theory is also circular. If a certain belief is true because it coheres with others, what do they cohere with? This is another example of an infinite regress. Also, since coherence theory is not a foundational theory, we cannot appeal to one or a select number of beliefs over the others, because all beliefs are equal.
3. What does coherence itself consist of? However, if someone were to establish criteria for coherence, this in itself would only be another believe, and so subject to the same criticisms.

There are two important properties relatively to consistency and coherency:

1. ***Consistency:*** $+A$ and $+\neg A$ cannot be both derived, unless they are already known as certain knowledge (facts)
2. ***Coherency:*** $+A$ and $+\neg A$ cannot be derived from the same knowledge base

2.9 Conflicts

[55],[56] In a multi-agent system, conflicts may arise because of two basic reasons: 1) different agents have contrasting *goals* 2) different agents have inconsistent *knowledge*. This situation can originate when agents are autonomous and

strongly motivated by their own interests, and when heterogeneous agents, with different skills, ‘histories’ and beliefs, coexist in a dynamic environment.

In a multi-agent environment, communication between agents becomes necessary when agents need to cooperate. If the system allows the agents to collect incomplete or uncertain information, or to adopt different goals, agents’ knowledge and/or beliefs may be inconsistent, and cooperation cannot be relied upon. Hence, any communication between agents which is aimed to organize the execution of a cooperative task will require an initial process of conflicts check and resolution, and the ability of an agent to understand and reason about another agent’s beliefs is vital for the success of this process.

This ability requires an agent to adopt and maintain an internal model of another agent’s knowledge and beliefs. As Grosz and Sidner have already pointed out, “any model (or theory) of the communication situation must distinguish among beliefs and intentions of different agents”.

Although research in modal logic has produced many theoretical results in the field of belief systems, the applications of these theories to other domains (such as conflictual multi-agent systems) have been, to a certain extent, neglected.

When a conflict (confrontation of divergent propositions) emerges from this interaction, it can be solved either in an epistemic way (focused on the task) or in a relational way (focused on the social comparison of competences). As for the effect of epistemic vs. relational conflict, it appeared that epistemic conflict is more likely to entail accuracy in problem solving and to produce long-term progress. For instance, when they observed interactions between children confronted to piagetian tasks, Carugati, De Paolis and Mugny (1980) and Mugny et al. (1978–79) noticed that only the participants who had solved the conflict in an epistemic way (namely, through confrontation of points of view) progressed durably. The benefit of the conflict was lost as soon as children showed compliance (relational regulation of the conflict).

Many studies have already explored the cognitive activities resulting from those two forms of conflict regulation, by creating situations requesting individuals to focus their attention either on the task or on social comparison.

3. Implementation

3.1 Systems Architecture

In our implementation, we consider the agents as nodes in a peer-to-peer network that adopts the *super-node* approach. Each node acts as an individual entity, performing its own actions, like sensing and reasoning, and maintaining its own local knowledge base. The agents are interacting with each other through the network by exchanging that knowledge. The internal knowledge is formed by rules, facts and constraints and the external knowledge from other agents is distributed as pieces of information which can be sensed.

There are two basic types of agents: ***Simple Agents (SAs)*** or agents & ***Master Agents (MAs)***. Simple agents are the majority in our system, which usually have lower capabilities for storing data and CPU. The simple agents are sensing local events, maintain their own knowledge base, perform their own reasoning process and distribute their knowledge about their environment variables to the agent system. A simple agent can also maintain locally protected knowledge, which is not distributed to the system. Each simple agent is connected to the nearest master agent, to whom the distributed knowledge is sent. Ordinarily, there is a simple agent for each smart device in our environment.

The master agents are the backbone of the system. They adopt the functionality of a super node. These agents support the system full communication among all agents and are responsible for the knowledge distribution. The master agents receive the local knowledge from the simple agents that are connected to them and publish it. In this version of the system there are not supported access policies for the distributed data. Each piece of information that a simple agent chooses to publish to the system, is globally available. Each master agent can communicate with the simple agents that are attached to it, and with its nearest master agents. A master agent could also control local variables. Specifically, a master agent can encapsulate a SA. The master agents have sufficient capabilities for storing data and CPU. The performance of the whole system depends on the analogy between the master and the simple agents (MAs/SAs), and the average number of simple agents that each master agent supports (average master agent effort).

A SA can be dynamically transformed into a MA and vice versa, according to the system's needs and the super node architecture. Also, agents can enter and leave the network dynamically. So, the final system becomes more flexible and efficient as it allows load balancing and redundant and thus fault-tolerant computation among different agents, leading to a more reliable system.

Each agent is independent and is asked to evaluate only a minimal number of constraints. They can use whatever software they have for this purpose. The cost of formalization for the whole system is reduced. Therefore the effort for maintaining and updating the system becomes manageable. As agents can work contemporary, parallel execution makes the entire process even more efficient.

In our implementation all agents are honest, perform absolute cooperation with all other agents and no agent possesses secret goals and aspirations, except from maintaining locally protected knowledge that is not distributed (like personal user data and preferences). Privacy is retained in such a way that agents only reveal information piecemeal when evaluating constraints and other agents only see information they are required to see for the solving process. No consideration has been taken for game theory issues between agents. The agents share common context representation and data definitions for the pieces of information that they distribute and commonly use.

JADE is an agent development tool, implemented in JAVA and FIPA-compliant. Peer-to-peer communication is supported. Although JADE provides all the mandatory components (FIPA) for the development of autonomous agents, it lacks the ability to include intelligent behavior to individual agents. Jess, a rule-based programming environment written in Java, provides a powerful tool for developing systems with intelligent reasoning ability. In [24], the cooperation of these two tools is presented, through the implementation of an intelligent business application.

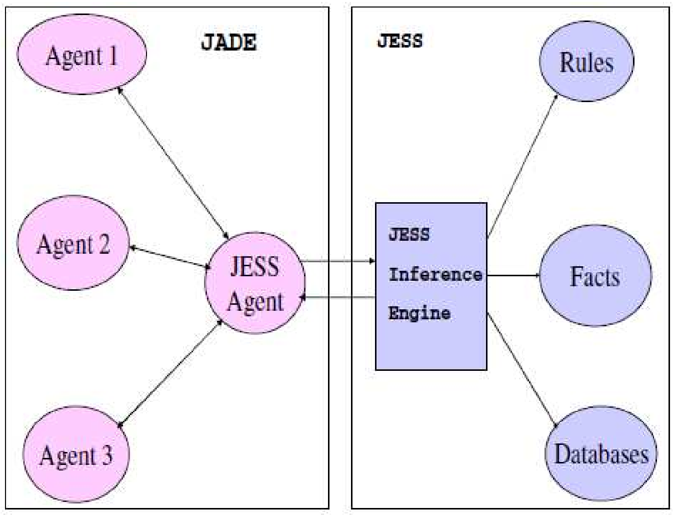


Figure 3-1: Integrate JADE and Jess

Respectively, for the agent system implementation we use JADE (Java Agent DEvelopment) framework. Each of these Jade agents is by default a single thread process, which implements a cyclic behavior for the reasoning process with GUI. The agents exchange ACL messages in order to achieve distributed knowledge.

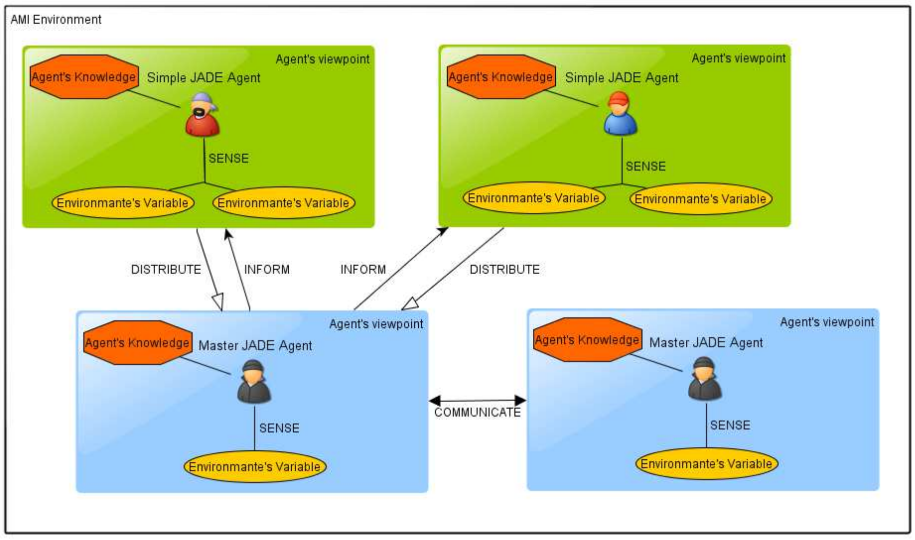


Figure 3-2: Agents' communication

For the reasoning process we use DEC (Discrete Event Calculus) [4], [5], [6] as the logic language for epistemic reasoning. We use DECKT (Discrete Event Calculus Knowledge Theory) [7], [8], [9], which is a Jess-based reasoner of DEC, implemented by Theodore Patkos (http://www.csd.uoc.gr/~patkos/index.htm). DECKT performs classical reasoning about DEC domain axiomatizations. Jess is a very efficient rule engine that employs the RETE algorithm for rule matching. We extend DECKT in order to distribute knowledge between agents.

We combine these tools by building an agent's behavior, embodied to each JADE agent, which performs intelligent reasoning for DECKT.

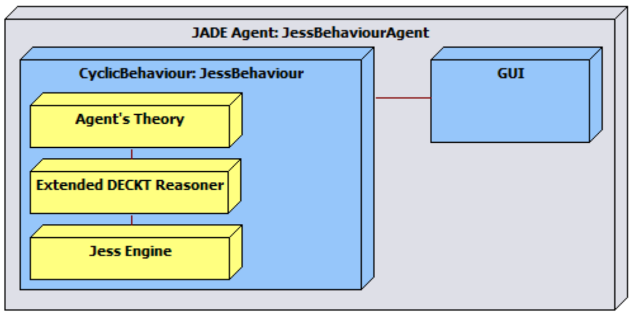

JADE Agent: JessBehaviourAgent
CyclicBehaviour: JessBehaviour
Agent's Theory
Extended DECKT Reasoner
Jess Engine
GUI

Figure 3-3: Software Layers

3.2 Agent's life cycle

The basic actions that an agent performs are reasoning tasks, sensing local events, communicating and distributing knowledge with other agents. Each Jade agent maintains a queue for the incoming ACL messages. When a new message is received, it is stored in the queue until the agent reads it. At the start up phase, the agent performs a reasoning task in order to calculate its initial state. Then the agent blocks until new events become known. When a new event arises, the agent adds the new piece of knowledge in its knowledge base and executes the reasoning process in order to calculate its new state. Those new events have the form of incoming ACL messages that contain valid DEC code statements.

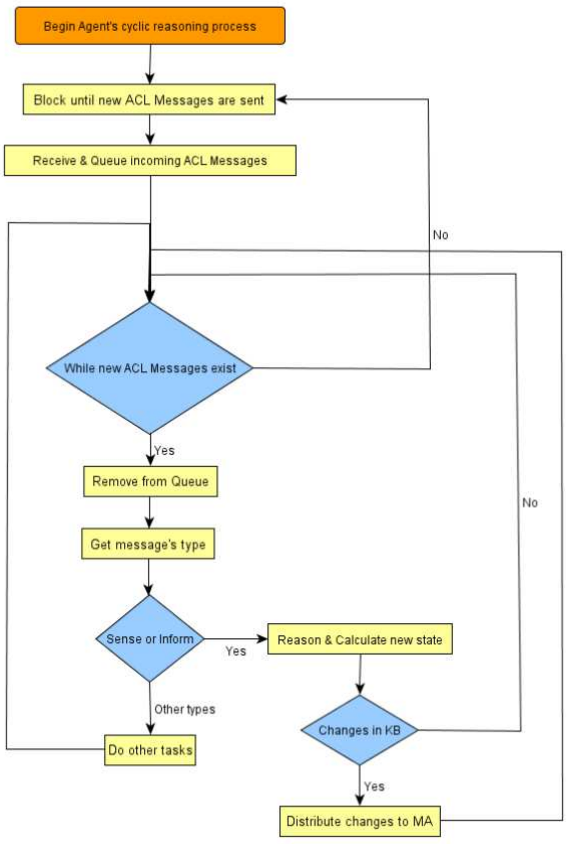

Begin Agent's cyclic reasoning process
Block until new ACL Messages are sent
Receive & Queue incoming ACL Messages
No
While new ACL Messages exist
Yes
Remove from Queue
No
Get message's type
Sense or Inform
Yes
Reason & Calculate new state
Other types
Changes in KB
Do other tasks
Yes
Distribute changes to MA

Figure 3-4: The agent's life cycle

The pieces of information that agents can sense and share, have the form of DEC code statements. Then, a simple parser is used and the DEC code is transformed in Jess code that is suitable input for the DECKT reasoner. The DECKT reasoner is executed and exports the models of the reasoning process. The final model is saved, as the current state of the agent, in the agent's Knowledge Base (KB). Finally, queries can be made to the KB to access that local knowledge.

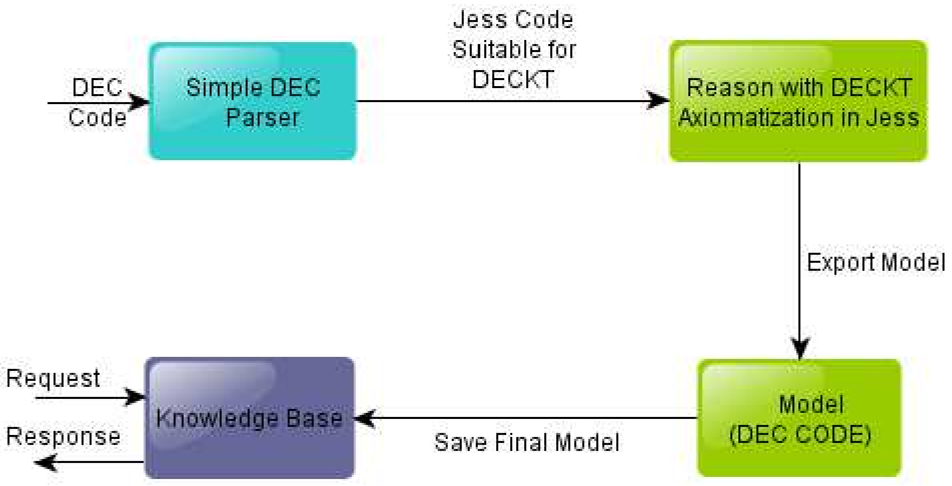


Figure 3-5: Information flow

The events that are originated by other agents are sent through the agents' network, until they are received and queued by the agent. The local events are sensed by the agent itself. In our demo, the user can type in a text area, the DEC code of the new events and by pressing the submit button, an ACL message will be constructed and queued for the agent. Every time that the reasoning process is repeated, the agent informs its master agent about possible changes of its local environment variables, and if there is no other task to be performed, the agent blocks. The master agent updates its knowledge base and publishes the new state.

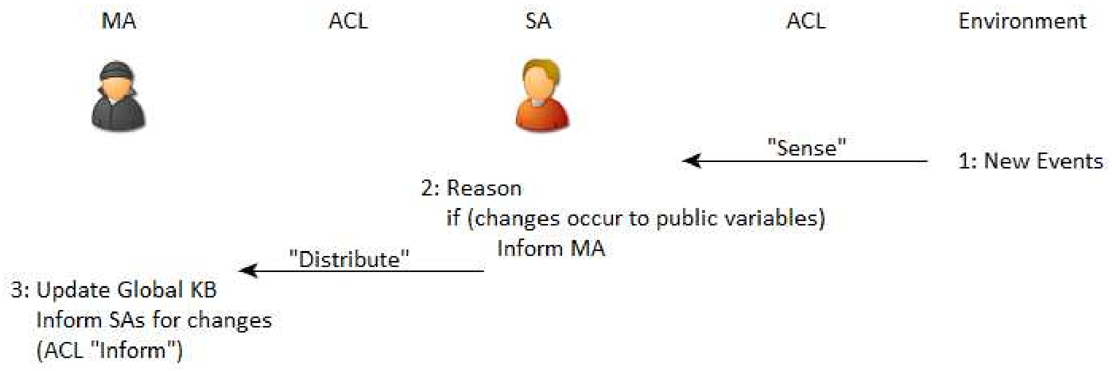


Figure 3-6: Agent's 'Sense' and 'Distribute' actions

The agent can respectively express its interest on specific environment variables that are maintained or are reachable by the master agent, in order to be automatically informed about changes on these variables. When the agent is not interested any more about these variables, it can inform the master agent to stop the informing procedure. (The agent can search about missing pieces of knowledge through its master agent by sending a specific message. Then the master agent will look up its own knowledge base and if no result occurs, it will carry forward the request to the other master agents.)

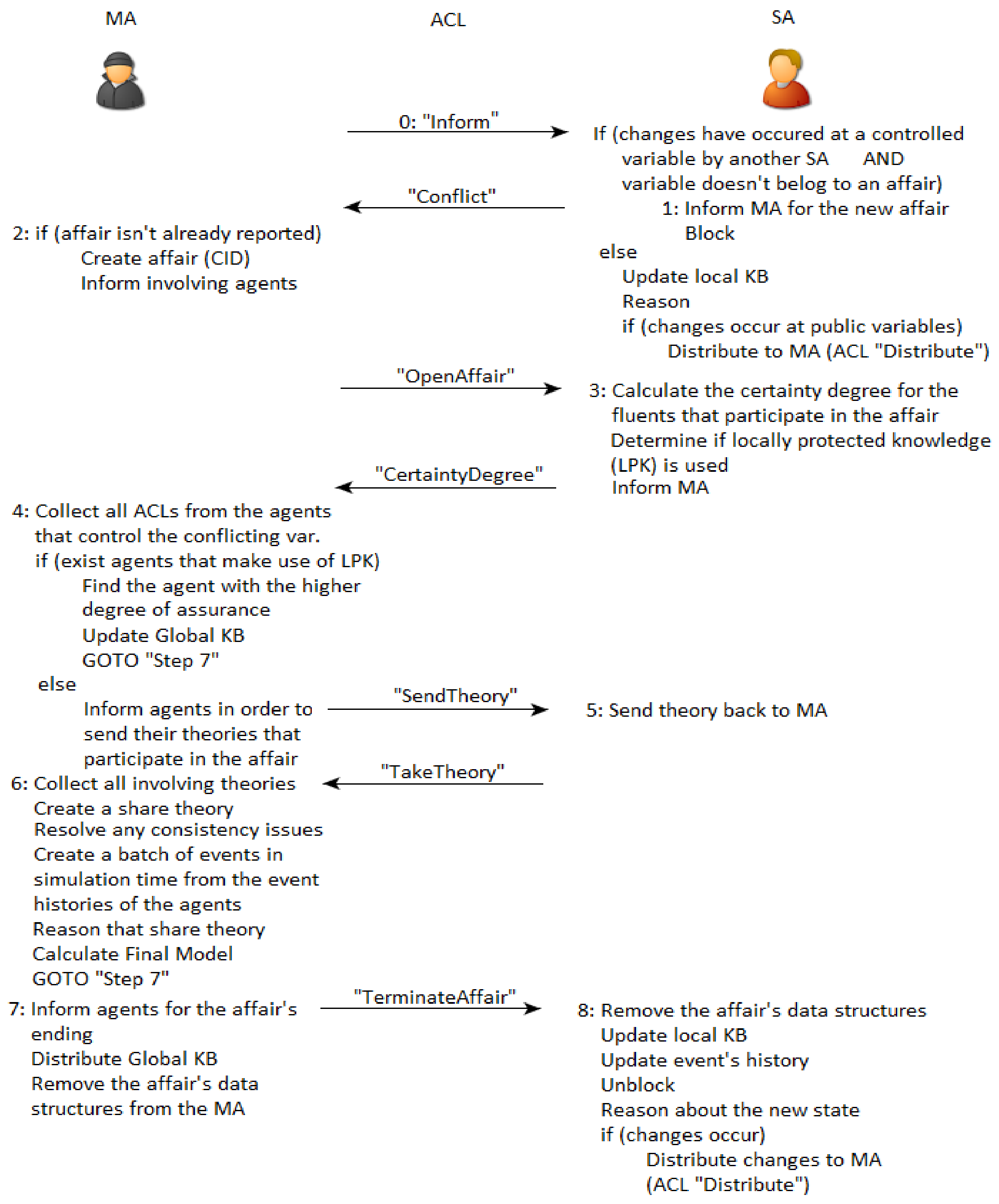
As agents distribute knowledge to the system, local conflicting conclusions can occur. In the ordinary way, each environment variable will be controlled by a single agent. But even by this policy, problems will happen. For example, if there are two devices in two different rooms that can recognize the presence of a person. What will happen if both devices result that Bob is recognized in their region. As Bob can't be in two rooms simultaneously, the agents that control those devices must communicate in order to resolve the affair. The involved agents will have to pass through a proof of evidence mechanism.

Through this proof of evidence mechanism, each agent is trying to convince all the others about his point of view. The agent uses the section of its theory that partakes in the conflict and the relative pieces of knowledge. The mechanism is composed by two main components, the ***share theory*** and the ***certainty degree***.

We introduce the *“share theory”* structure. Each involved agent contributes in the share theory’s construction by the portion of its local theory that contains the conflicting set of variables. A share theory is the combination of many such local theories, where the duplicate rules have been removed. The agents publish their theories that are involved in this process, and a shared theory is constructed in a MA which will determine the state of the variables. Along with its theory, each agent will also send a small history of the events that influence the variables. The MA will sort the events from all agents by the real time that they occur. The events will be used in the reasoning process with the shared theory and the final model will be calculated.

The *“certainty degree”* is a recommended grading mechanism. In order an agent to be able to reason about the state of a variable, there must be a set of rules, which contain the variable, that fire. The supporting rules add a positive value for the certainty degree calculation. The counter rules add negative values. The summation of all these values for all rules is the *theory contribution* (TC) of the agent for the specific variable. There can also be implemented policies where combination of rules can produce different results. For example if only r1 or only r2 fired they could add +1, but if they fired simultaneously they could add -1. The TC will take values in the space between 0 – 10. A value of 0 – 5 will represent normal degree of evidence about the state of the examined variable. A value of 6 – 10 will represent strong evidence about the state of the examined variable. There is also the *local knowledge contribution* (LKC). The LKC is taking a value from 1 to 5 if the agent uses its locally protected knowledge for the specific reasoning process, otherwise it is 0. The locally protected knowledge is not distributed to the system and is manipulated and maintained only by this agent. We suppose that these pieces of knowledge are either related to variables of local interest that are not distributed due to efficiency issues or are related to agent’s privacy information, like security and user’s data. The use of privacy information is considered as more valuable and contributes with greater values in the LKC. It is noticed that those pieces of protected knowledge that are not ordinary distributed due to efficiency issues, can be embodied to a share theory structure. The final certainty degree is the sum of TC and LKC. Each agent calculates automatically the certainty degree of each variable during the reasoning process. When this feature is used for a conflict’s resolution, each involved agent publishes its certainty degree for the set of conflicting variables along with the states of these variables to the MA that is responsible for the affair. The MA will determine the final state for these set of variables according to the information that is distributed by the agent with the greater certainty degree. All agents will adopt his point of view. Apart from the certainty degree, the MA can take into consideration the agents’ roles or hierarchy (i.e. in case of tie).

If each phase fails to determine the current global knowledge state we can exclude the set of variables that cause the uncertainty and compute the current global knowledge state with the remaining variables, keeping no knowledge about the conflicting variables.



```
If (changes have occured at a controlled
    variable by another SA      AND
    variable doesn't belog to an affair)
        1: Inform MA for the new affair
           Block
else
    Update local KB
    Reason
    if (changes occur at public variables)
        Distribute to MA (ACL "Distribute")
```

```
2: if (affair isn't already reported)
        Create affair (CID)
        Inform involving agents
```

```
3: Calculate the certainty degree for the
   fluents that participate in the affair
   Determine if locally protected knowledge
   (LPK) is used
   Inform MA
```

```
4: Collect all ACLs from the agents
   that control the conflicting var.
   if (exist agents that make use of LPK)
        Find the agent with the higher
        degree of assurance
        Update Global KB
        GOTO "Step 7"
   else
        Inform agents in order to
        send their theories that
        participate in the affair
```

```
5: Send theory back to MA
```

```
6: Collect all involving theories
   Create a share theory
   Resolve any consistency issues
   Create a batch of events in
   simulation time from the event
   histories of the agents
   Reason that share theory
   Calculate Final Model
   GOTO "Step 7"
```

```
7: Inform agents for the affair's
   ending
   Distribute Global KB
   Remove the affair's data
   structures from the MA
```

```
8: Remove the affair's data structures
   Update local KB
   Update event's history
   Unblock
   Reason about the new state
   if (changes occur)
        Distribute changes to MA
        (ACL "Distribute")
```

Figure 3-7: The Proof of Evidence Mechanism

3.3 Certainty degree, share theory and other approaches

We introduce the share theories as the basic technique to resolve affairs. The certainty degree calculation is introduced in the proof of evidence mechanism in order to enable us resolving an affair fast, without the need to create the share theory, and in cases where we encounter reasoning with locally protected knowledge about security or privacy data. Other approaches have been also used to other systems, like *grading mechanisms* that use the Bayesian Law and the Kappa calculus. All those approaches are trying to calculate how likely or unlikely a possible world is. However, like the certainty degree calculation, problems can occur if the agents' theories aren't constructed carefully. The features that are used in order to give 'points' to a possible world are usually subjective, like an approach that assigns greater gradation for agents that are close to the elements that cause the affair.

> "Consider a scenario where a *room agent* inferences that the room's lights are on, when someone turns on the lights' switch. The *building agent* knows if the building has electric power or not and also possesses the constraint which indicates that a room's lights can be turned on only if the building has electric power. What will happen if someone turns on the lights' switch when the electric power is off?"

If the agent's opinion, that is closer to the affair's cause, is considered as the most correct, then we would falsely infer that the lights are turned on. These kinds of faults could be avoided if we directly use the share theory technique. In that case, the share theory would contain the room agent's rules about the lights, the building agent's constraint about lights and the knowledge for both agents that the building has no power and that someone has turned the lights' switch on. The share theory would correctly infer that the lights aren't turned on.

Besides the complexity of creating a share theory in a dynamic evolving environment, there are cases where a share theory could also generate wrong results. Faults would occur when locally protected knowledge affects an agent's behavior about an affair.

> "Consider a scenario where an agent desires to inform Bob about a tennis match that will take place next week. Another agent that keeps Bob's profile as locally protected knowledge would know that Bob is not interested about tennis. This agent would try to prevent the first one from annoying Bob, but is not willing to distribute Bob's personal data to contribute at the share theory's construction."

As the second agent can't argue about his point of view about the affair, the first agent will inform Bob. In that case a grading mechanism, like the certainty degree, could have resolved the affair correctly.

So if agents' theories can't be rated well by a certainty degree, the feature should be avoided and a share theory should be created every time conflicts occur. Yet the certainty degree should be used in cases that a share theory can't resolve. For our case study, we use both mechanisms. We use the certainty degree approach only when we must encounter reasoning with locally protected knowledge and the share theories in all other cases.

When we construct a share theory, we consider each of the local theories that apart it, as independently consistent. Thus, we could expect that the share theory would be globally consistent but this is not always the case.

> “Consider a scenario where two agents are involved in an affair for ‘a’. The agents contribute with the following theories:
>
> | SA1 | SA2 | Share Theory for ‘a’ |
> |---|---|---|
> | r1: $b \rightarrow a$ | r2: $b \rightarrow \neg a$ | r1: $b \rightarrow a$ |
> | HoldsAt ‘b’ | | r2: $b \rightarrow \neg a$ |
> | | | HoldsAt ‘b’ |
>
> The share theory is inconsistent. ”

In our study, we assume that the theories developers will keep both local and share theories consistent but we expect that such matters would arise in our system.

Bikakis [1] has proposed a distributed reasoning algorithm with conflicts for a P2P system, where such problems in *distributed theories* are encountered by the use of *trust information* among the system’s nodes. Thus, the rules of more trusted nodes are taking into account in these cases and the consistency is restored. We can expand our system to support such policies among agents by taking advantage of the rules’ priorities and preferences mechanism that is presented below in the reasoning process chapter.

In the same way, we implement a mechanism that takes into consideration the agents’ roles or hierarchy. The features of agents’ roles and hierarchy are very common in multi-agent applications like AMI. As we expect that conflicts will occur between agents that are associated with some kind of social relation, we believe that we could effectively classify them according to their local roles or hierarchy. Considering the conflict scenario about the lights condition between the room and the building agent, we could assume a topology where the building agent would be higher in the hierarchy or a social agent network where the building agent would play a significant role, like the administrator and the controller of the other agents inside the building. In such cases, the rules that are originated by the building agent would be of greater priority and if consistency matters would occur during the share theory’s creation, these rules would be taken into consideration to restore the theory’s consistency.

Both approaches can resolve the inconsistency matters of a share theory but we can still come to false conclusions about the global state of the system. In the example above, the room agent could ultimately be the right one. However, the system will remain globally coherent in all cases.

These approaches could also be used for conflict resolution in cases where there is more than one agent that participate in an affair and who take use of their locally protected knowledge, making the certainty degree and the share theory features incapable for resolving that affair. Thus the knowledge of agents who are more trusted or higher in the hierarchy or play a significant role could be subsidized.

In our demo system implementation, the categorization of agents in SAs and MAs is related to the network operation and not with the agents' sociability. Also no information is kept about trustworthy agents. However, we maintain a hierarchy structure. The demo implementation retains the basic proof of evidence mechanism as it is described in the 'Proof of evidence mechanism algorithm' section below.

3.4 Causality and Distributed Knowledge

The agents publish their local knowledge in the system and express their interest on variables that are maintained or are reachable by other agents, in order to be automatically informed about changes. Thereby, anything an agent knows about the other agent's is distributed knowledge to the agents that can causally affect it. This kind of interaction composes a causality network among the agents and corresponding causal structures. The [10] deals with distributed knowledge and information flow in causality asynchronous systems and clarifies the intuition that the causal structure of a system, constraints agents to acquire only external knowledge that is distributed knowledge to the agents that may causally affect it.

3.5 The Reasoning Process

3.5.1 Introduction

The event calculus can support reasoning tasks for event that happen sequentially or parallel. The DECKT extends the basic reasoning process with HCDs (Hidden Casual Dependencies), a mechanism that allow us to perform reasoning when there are missing pieces of knowledge which may become know in a future time.

> "As an example, we can consider the lights in a room. When we press the switch we expect that the lights will open if the room has electric power. If we press the switch at t=0 but we have no knowledge about the electric power we have no clue about the room's lights. But if in a future time we learn that in t=0 the room has electric power, and no changes have been made so far, we can conclude that the lights from time point 0 until now, remain open."

The classical reasoning in EC, simulates the reasoning process for virtual time points (like t=0,1,2,…). We extend the reasoning process in order to support events with real duration when they occur at runtime.

> "As an example, we can consider Bob's glassfish, which must be fed twice a day. From one feeding to the other, there must be at least ten hours and at most twelve hours elapsed time."

Every time that the reasoning process is taking place, a new Jess engine is constructed for the DECKT's reasoner, and the reasoning in DEC is performed for time points from 0 to *end time*. Then the reasoner stops. The end time of a reasoning process is estimated dynamically at run time and represents the last time point for which the reasoner will calculate a model. This final model that is produced represents the current state of the agent's local knowledge base. When a new event arises, we add its corresponding code representation to the reasoner and the process is repeated, noticing that the agent's knowledge base 'holds' for the relevant time point 0. The new events should take account that the reasoning process is always starting from 0.

3.5.2 Events with duration

One of the main problems of expression is time. Not every indirect controlled action is to be executed immediately. Some contexts trigger special situations in which the required action depends on some time factor, e.g. "Turn off the oven in 20 minutes" or "if somebody enters the room in the next 5 minutes send me a message". In order to implement a system that is operating in real time, such issues must be encountered.

In our system, each agent can support two types of time points. The first is the relevant discrete time points that the EC simulation process provides for a batch of EC statements. The second type is the use of real system time (hours : minutes : seconds).

For the time duration event, a mechanism with Java Timer objects is used. Each agent can maintain a hash table with Timer objects. For each time point where an event with duration must occur the agent creates a timer that fires when the specific time elapsed. The key of the hash table is the different time points. Each slot contains the Timer and the list of rules that cause the Timer's creation. A rule's effect on an event with duration can be cancelled in a future time point before the Timer fires. So a rule can deleted from a Timer's list. When all rules have been deleted from a Timer's list before it fires, the Timer must be deleted too. When a timer fires an empty ACL message is sent to the agent and the Timer is destroyed. The message is added to the agent's message queue. When the agent detects the message, it will start the reasoning process. It's up to the agent's rule base developer to manipulate these kinds of events. When such an event occur, a new fluent must be created which will represent the actual time that the event occur.

> "For example, for the fluent WasFed(PetName) that represents the knowledge that the pet with PetName was fed, there must be created the fluent TimeDurationWasFed(PetName,hout,min,sec). Then the developer must implement rules that will deal with this new fluent. "

This is one of the main features that are presented in this thesis. The mechanism that is described above enable us to reason about real time events. We achieve all the reasoning capabilities of EC for real time points. Parallel and sequential occurrences of events can be handled for system time. Conditional statements like "check if $t_1 > t_2$", "check when *system time* >= *t*" or "check when for the nth occurrence in the space $t_1 > t_2$ " can be also modeled. Such statements are very common in real applications like economic and industrial systems and AMI systems. The mechanism allows us to implement assisting living scenarios like the following one:

> "Bob is sick and must stay at home this week. We want to attend Bob's medication. He has to take a pill three times per day. From one dose to the other, there must be at least six hours and at most eight hours elapsed time."

Our approach is quite efficient since the agent blocks in the idle time and operates only when the reasoning process must be executed to calculate its new state. The payload of the whole mechanism remains manageable even for large amount of data.

Ordinary, the system time is taken account once when the reasoning process begin and remains the same until the process ends. When we create a share theory, we use the real system time points that are reported to the sorted events' history, which vary in every round of the reasoning process and may differ from the current system time.

The latest version of DECKT is following another policy for confrontation of real events which are happening at real time. Every time that a reasoning process is performed, the reasoner pauses when it reaches the end time. When a new event arises, the reasoning process continues from that time to a new end time which can be assigned dynamically. The new event is assigned a time point after the previous end time where the reasoner was paused. This approach solves the problem of manipulating events that are happening at real time but it cannot manipulate events with real time duration like the suggested one. There are also some drawbacks like the number of facts that are created by the Jess engine, as reasoning processes are executed, and have to be usually flushed. Besides that discrete time points could evolve to extremely large numbers, if this version is used for a real system, that could even cause a number's overflow.

Other approaches use global system time, which is specified by some *time agent* in the network. By contract, the role of this agent is to maintain a global system time that will be adopted by all the other agents. In that case, issues like time distribution to agents and synchronization must be considered.

In his PHD dissertation, Manuel Garcia [19] deals with intelligent environments too, adopts the Event Condition Action (ECA) rule language and the blackboard representation and proposes a mechanism, similar to our approach. He created the *Timer*, a special action, executed as a new thread, with an ending horizon, an optional set of rules to execute when it started, a set of rules to analyze and execute while the thread is running and an extra set of rules to execute when it is finished. Since some actions would be executed some time after they were decided to be taken, a mechanism for turning back the decisions or changing the time horizon was added. Thus the on-running rules may affect not only the world but also the timer thread itself or use the state of the thread as part of the context through the following actions:

- Timer.*pause*
- Timer.*play*
- Timer.*stop*
- Timer.*kill*
- Timer.*start*
- Timer.*replay*
- Timer.*reset*

Our mechanism with the Timer objects can achieve similar functionality for the Timers that are created. We simulate the *start* function by the ordinary way that a Timer is operating. An existing Timer is deleted before it fires when all rules that cause its creation, are deleted too, simulating the *stop* and *kill* functions. We can *reset* a Timer when we delete it and create a new one by maintaining the Timer's data and setting the new fire point. We can make a Timer to *replay* its action if we make use of its 'delay' parameter and 'setRepeats' method during the Timer's initialization. We can *pause* and *play* a Timer if we use the methods 'stop' and 'restart'.

This approach enables us to deal with several cases of real world states and empowers the expressiveness of our tool. Consider the following scenario that make use of the start, replay and stop actions:

> "Bob sets the alarm clock of his smart phone to be activated at 8 o'clock the next morning. Bob deactivates the smart phone before he fell asleep. At 8 o'clock the alarm is triggered. Bob stops the alarm. If Bob doesn't turn on his smart phone, the alarm will be activated every 5 minutes. When Bob turns on his smart phone, indicates that he is aw and the alarm is deactivated."

When Bob sets the alarm the Timer is created to fire (play) at 8 o'clock. After the first activation the Timer repeats its action (replay) every 5 minutes. When Bob turns on the smart phone, the Timer is destroyed (stop, kill).

3.5.3 Rules' priorities and preferences

Each constraint that is declared with DEC is translated in Jess rules. Each Jess rule posses a "salience" variable. The value of the salience can determine the order in which, the rules that should fire simultaneously, will eventually be evaluated. Thus, we can introduce rules' priorities and preferences in the reasoning process.

> "For example, Bob is receiving a video call and we want this call to be served by some device. The device must be in same room as the Bob. If more than one device exists in the room that can serve a video call, we can choose the most preferable one. If no such device exists in the room, we choose the default device that serves video calls (a smart phone). "

The larger the salience value, the largest the priority of the rule. Thus, by assigning high values we can express high preference. When the video call is attach to a specific device, a fact in Jess engine can be asserted in order to prevent the lower priority rules to fire.

According to the possible worlds representation, in such cases there are several possible worlds and we choose the one, which is more suitable for us, as the current state, based on our subjective information. For the example above, there would be as many possible worlds as many the devices that serve video calls are. In each of these worlds, a different device would serve Bob's call. But we choose only one of them and reject all the others based on our preference policy.

This technique enriches the reasoning process by giving the capability to a developer to express preference among the different probable states. Users' preferences can be modeled in order to achieve better performance for AMI systems. Besides it enable us to express priorities for the rules' execution sequence. We could embody this feature in the proof of evidence mechanism or use it to resolve conflicts as it is described in the following sections.

The theory's consistence is up to the theory's developer. We use this capability when we know that an event has happened and we want to determine which the result will be. That is, the event happens but no effect has been occurred yet. Like the example scenario, the agent receives a new video call request, but still no device has served the call. Then the agent determines which device will serve the call. We cannot use this capability in order to reason what has actually happened when an event has occurred. For example if another agent is informed about the new video call request, he cannot determine which is the device that actually serves the call according to his own preferences, as he is not responsible for this call administration. He must be informed by the first agent about the final state.

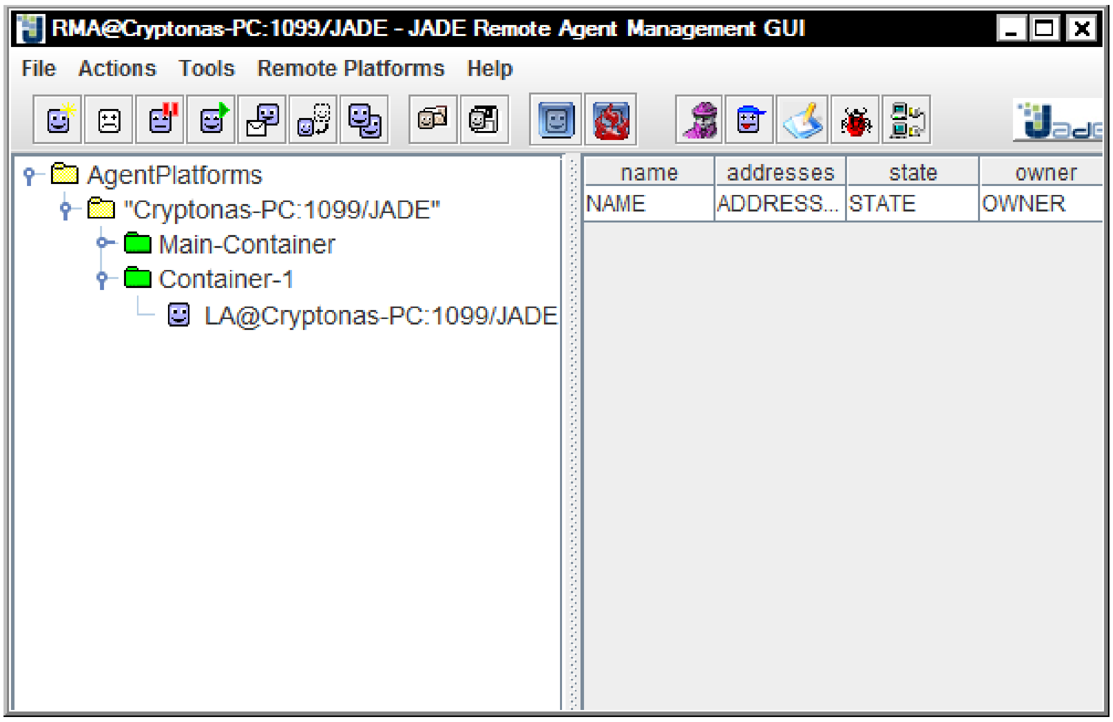
3.5.4 Proof of evidence mechanism Algorithm

Every time where conflicts are detected during the distributed knowledge integration, we pass through the proof of evidence mechanism:

```
Step 1 (SA) : if ( changes to a controlled variable occur by another agent ) then
                  Inform MA for the affair
                  Block until response is received
Step 2 (MA) : if ( affair message is received ) then
                  If ( affair is not already reported ) then
                          Create a unique ID (CID)
                          Inform involving agents about the affair (send CID)
Step 3 (SA) : if ( affair is reported ) then
                Calculate the certainty degree for the fluent that participate in the affair
                Determine if locally protected knowledge is used
                Inform MA
                Block
Step 4 (MA) : Collect all ACL's from agents that control the conflicting variables
                If ( agents exist that make use of LPK for privacy or security data ) then
                  Find the agent with the higher degree of assurance
                  Update KB
                  GOTO 'Step 7'
                Else
                  Inform agents to send their theories that participate in the affair
Step 5 (SA) : Send theory to MA
Step 6 (MA) : Collect theories
              Create a shared theory from those theories
              Resolve any consistency issues
              Create a batch of events in simulation time from the events histories of
the agents
              Calculate the final model
              Update KB
              GOTO 'Step 7'
Step 7 (MA) : Inform agents for the affair's ending
              Distribute KB
              Remove the affair's data structures
Step 8 (SA) : Remove the affair's data structures
             Update local KB
             Update event's history
             Unblock
             Reason about the new state
             If (changes occur) then
                 Distribute changes to MA
```

3.5.5 Agent's GUI

We use Eclipse IDE for the development. Eclipse comes up with eJADE, a plug-in for JADE that helps agent developers to launch the agent platform and to deploy agents, and JessDE (Jess Developer's Environment), a plug-in for Jess including an editor, a debugger and a Rete network viewer. In the figures below, we have deployed the 'LA' (Living room Agent) agent of a smart home.


RMA@Cryptonas-PC:1099/JADE - JADE Remote Agent Management GUI

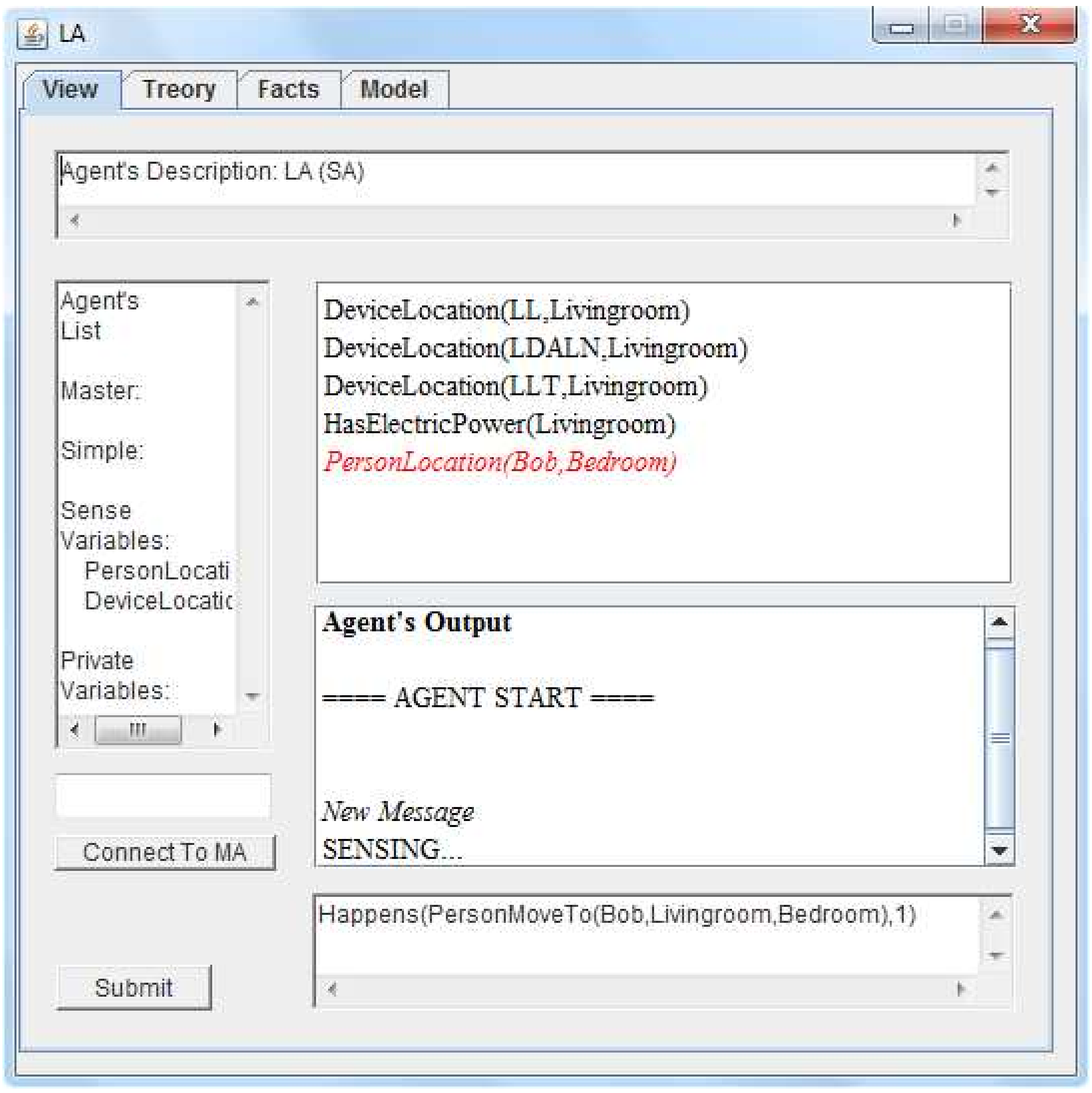
File Actions Tools Remote Platforms Help

- AgentPlatforms
  - "Cryptonas-PC:1099/JADE"
    - Main-Container
    - Container-1
      - LA@Cryptonas-PC:1099/JADE

name	addresses	state	owner
NAME	ADDRESS...	STATE	OWNER

Figure 3-8: JADE Remote Agent Management GUI

The agent's GUI contains four tabs. The first one is the agent's ***view*** and contains the agent's information about the environment. This is the most basic tab. At the top of it, there is the agent's name and its type (SA or MA). At the left, there is a text area which contains information about the Master Agent and the Simple Agents, the variables that are sensed by the agent and the variables that are locally protected knowledge for the agent and are not distributed. The LA agent senses the variables 'PersonLocation' and 'DeviceLocation' and has no private variables. In the center there is the agent's *knowledge base* (KB). With red color are denoted the last changes of the KB. In this example, the LA has just sensed the event that 'Bob has moved from the living room to bedroom' and Bob's location has changed. Below the KB area there is the agent's *output*, where messages about the reasoning process are printed. There are also two input areas. In the first one, the user can enter the MA's name to who the agent will be attached. In the second one, the user can type DEC code that represents the local events that the agent will sense.



Agent's Description: LA (SA)

Agent's
List

Master:

Simple:

Sense
Variables:
PersonLocati
DeviceLocatic

Private
Variables:

DeviceLocation(LL,Livingroom)
DeviceLocation(LDALN,Livingroom)
DeviceLocation(LLT,Livingroom)
HasElectricPower(Livingroom)
PersonLocation(Bob,Bedroom)

Agent's Output

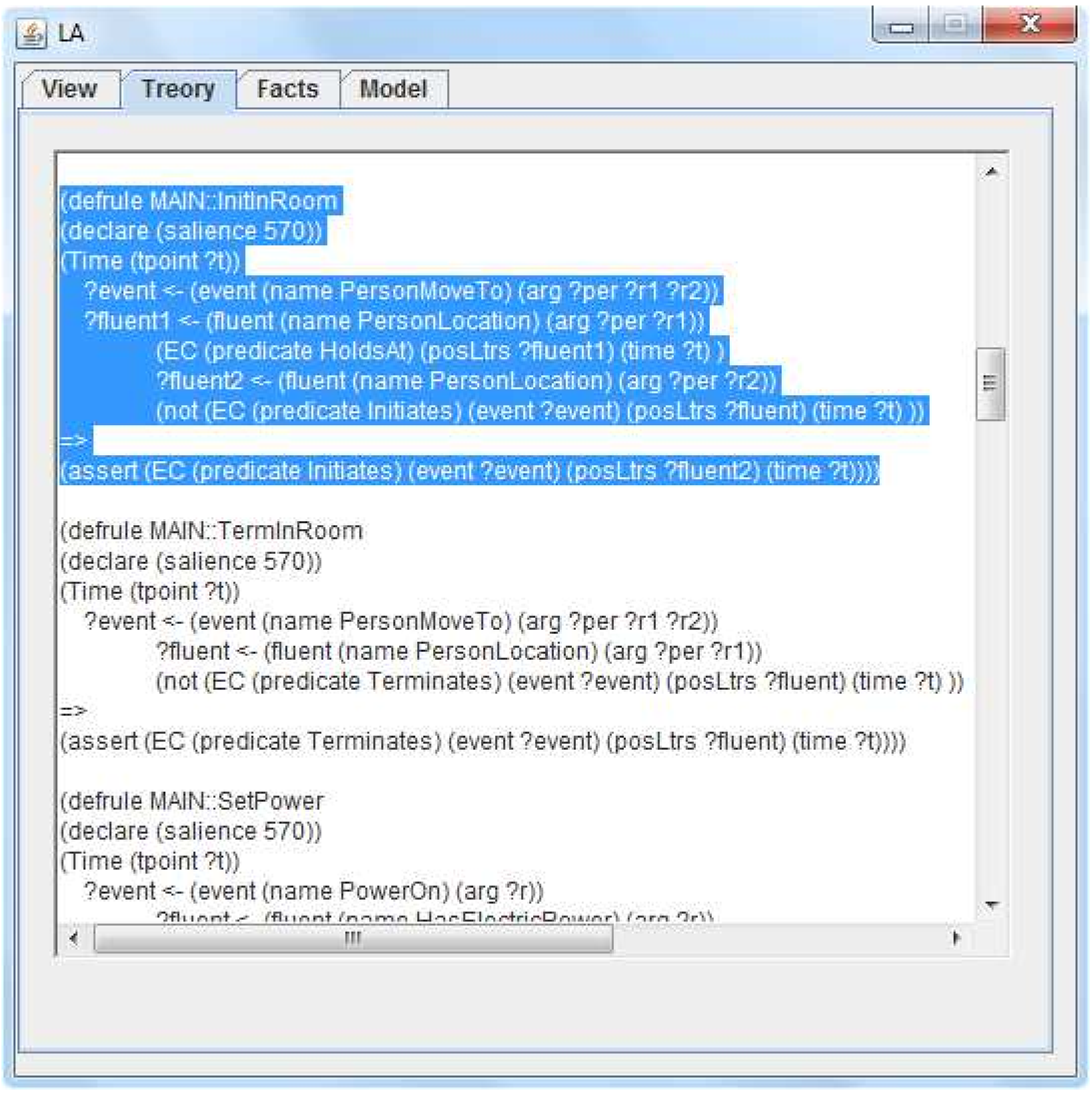
==== AGENT START ====

New Message
SENSING...

Happens(PersonMoveTo(Bob,Livingroom,Bedroom),1)

Figure 3-9: Agent's 'view'

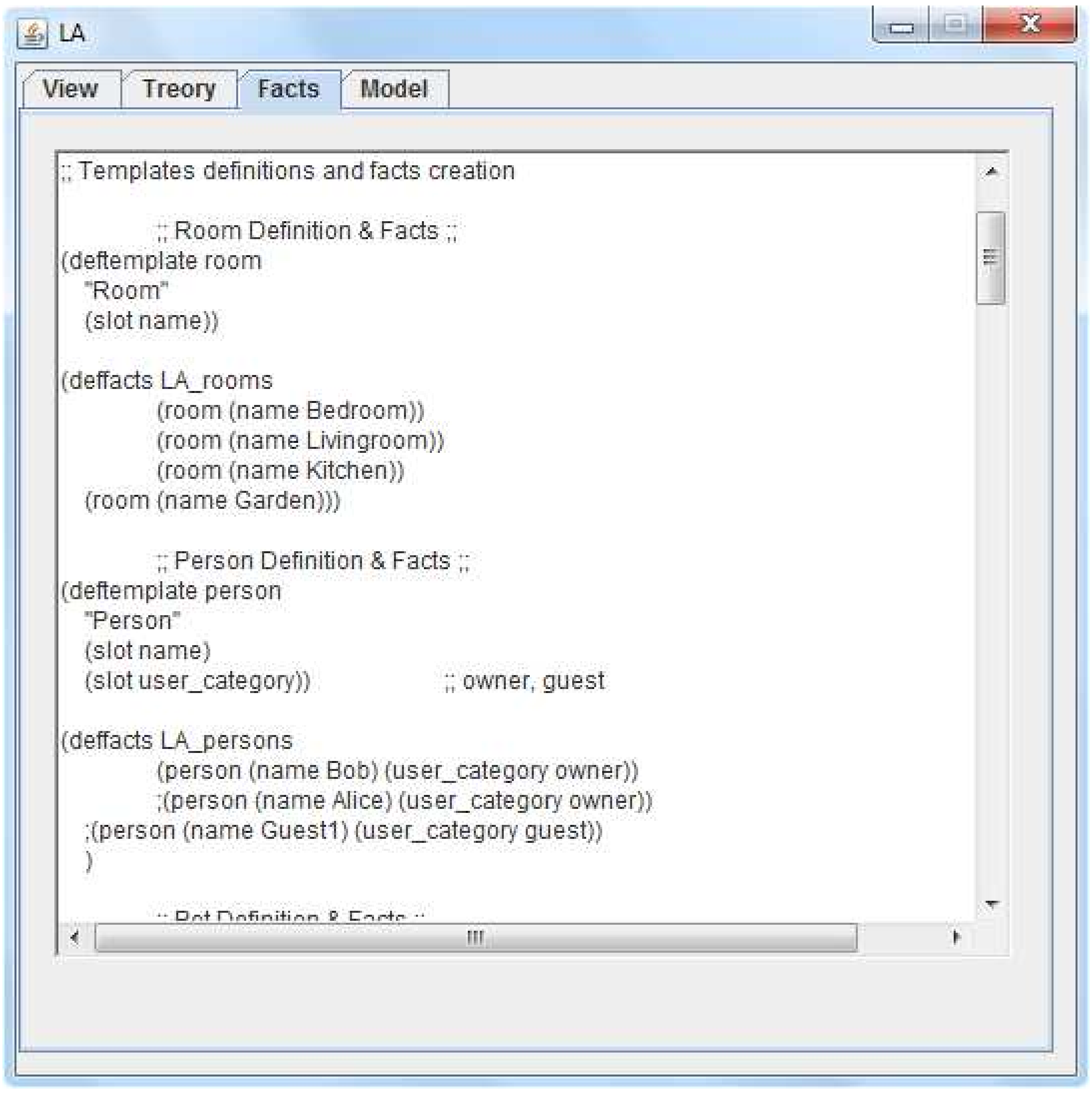
The second tab contains the agent's theory. The theory is composed by rules and constraints.



```
(defrule MAIN::InitInRoom
(declare (salience 570))
(Time (tpoint ?t))
   ?event <- (event (name PersonMoveTo) (arg ?per ?r1 ?r2))
   ?fluent1 <- (fluent (name PersonLocation) (arg ?per ?r1))
         (EC (predicate HoldsAt) (posLtrs ?fluent1) (time ?t) )
         ?fluent2 <- (fluent (name PersonLocation) (arg ?per ?r2))
         (not (EC (predicate Initiates) (event ?event) (posLtrs ?fluent) (time ?t) ))
=>
(assert (EC (predicate Initiates) (event ?event) (posLtrs ?fluent2) (time ?t))))

(defrule MAIN::TermInRoom
(declare (salience 570))
(Time (tpoint ?t))
   ?event <- (event (name PersonMoveTo) (arg ?per ?r1 ?r2))
         ?fluent <- (fluent (name PersonLocation) (arg ?per ?r1))
         (not (EC (predicate Terminates) (event ?event) (posLtrs ?fluent) (time ?t) ))
=>
(assert (EC (predicate Terminates) (event ?event) (posLtrs ?fluent) (time ?t))))

(defrule MAIN::SetPower
(declare (salience 570))
(Time (tpoint ?t))
   ?event <- (event (name PowerOn) (arg ?r))
```

Figure 3-10: Agent's 'theory'

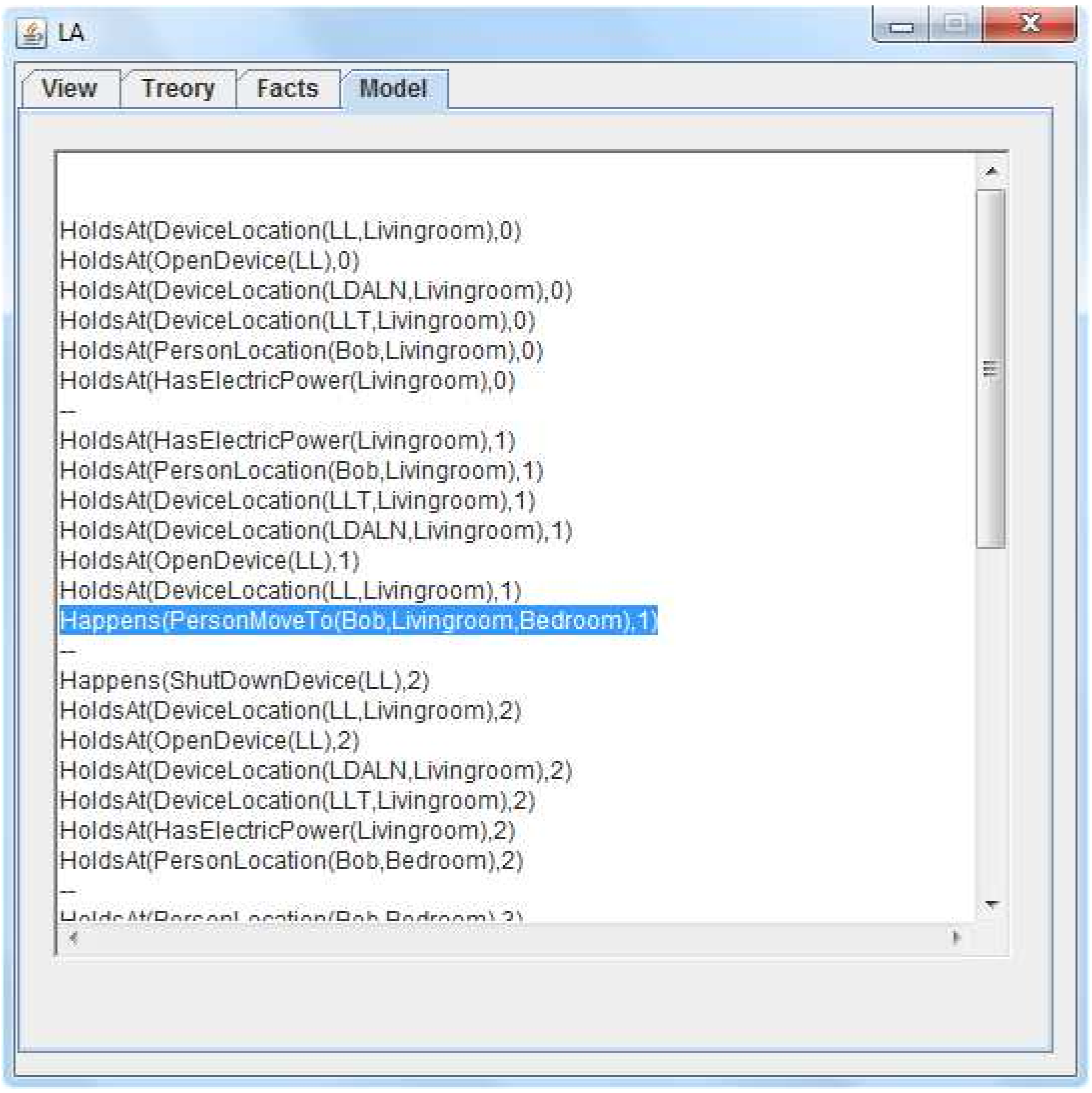
The third tab contains the agent's facts. There are the basic definitions and facts declarations.



```
;; Templates definitions and facts creation

            ;; Room Definition & Facts ;;
(deftemplate room
   "Room"
   (slot name))

(deffacts LA_rooms
            (room (name Bedroom))
            (room (name Livingroom))
            (room (name Kitchen))
   (room (name Garden)))

            ;; Person Definition & Facts ;;
(deftemplate person
   "Person"
   (slot name)
   (slot user_category))                ;; owner, guest

(deffacts LA_persons
            (person (name Bob) (user_category owner))
            ;(person (name Alice) (user_category owner))
   ;(person (name Guest1) (user_category guest))
   )
```

Figure 3-11: Agent's 'facts'

The fourth tab contains the agent's model. The DECKT's output models are printed in this tab every time the reasoning process is performed. Here are the agent's models for the event that was previously sensed (PersonMoveTo).



```
HoldsAt(DeviceLocation(LL,Livingroom),0)
HoldsAt(OpenDevice(LL),0)
HoldsAt(DeviceLocation(LDALN,Livingroom),0)
HoldsAt(DeviceLocation(LLT,Livingroom),0)
HoldsAt(PersonLocation(Bob,Livingroom),0)
HoldsAt(HasElectricPower(Livingroom),0)
--
HoldsAt(HasElectricPower(Livingroom),1)
HoldsAt(PersonLocation(Bob,Livingroom),1)
HoldsAt(DeviceLocation(LLT,Livingroom),1)
HoldsAt(DeviceLocation(LDALN,Livingroom),1)
HoldsAt(OpenDevice(LL),1)
HoldsAt(DeviceLocation(LL,Livingroom),1)
Happens(PersonMoveTo(Bob,Livingroom,Bedroom),1)
--
Happens(ShutDownDevice(LL),2)
HoldsAt(DeviceLocation(LL,Livingroom),2)
HoldsAt(OpenDevice(LL),2)
HoldsAt(DeviceLocation(LDALN,Livingroom),2)
HoldsAt(DeviceLocation(LLT,Livingroom),2)
HoldsAt(HasElectricPower(Livingroom),2)
HoldsAt(PersonLocation(Bob,Bedroom),2)
--
HoldsAt(PersonLocation(Bob,Bedroom),3)
```

Figure 3-12: Agent's 'Model'

3.5.6 Messaging

We summarize the number of messages that are required for each action in our system. 'N' denotes that number of SAs that participate in the action. We can deduce that the system's performance is propositional to the number of the SAs that participate.

Action	Number of Messages
Sense	**1**
Sense & Distribute	**2**
Inform	**N**
Inform, Conflicts & Certainty Degree Calculation	**N** *(Inform SAs)* + **1** *(report conflict)* + **N** *(ask for certainty degrees)* + **N** *(the certainty degrees)* = **3N+1**
Inform, Conflicts & Share Theory Creation	**N** *(Inform SAs)* + **1** *(report conflict)* + **N** *(ask for individual theories)* + **N** *(the individual theories)* = **3N+1**
Inform, Conflicts & Full Proof Of Evidence Mechanism	**3N+1** *(Use of Certainty Degree)* + **N** *(ask for individual theories)* + **N** *(the individual theories)* = **5N+1**
Affair's Termination	**N** *(Terminate Affair)* + **N** *(Inform the new state)* = **2N**

Table 3-13: Messages per action

4. Evaluation

4.1 Virtual Smart Home

In order to present our multi-agent system, a virtual smart home has been implemented. In this house there are three main rooms in a row:

- Bob's bedroom
- The living room
- And the kitchen

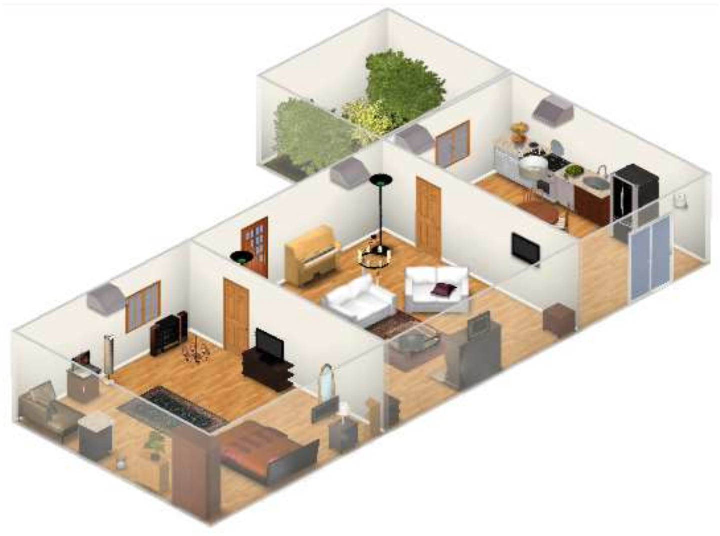


Figure 4-1: The Virtual Smart Home

In each room there are some smart devices, like:

- Bedroom's smart devices
  - TV
  - PC
  - Lights
  - Bob's smart phone
- Living room's smart devices
  - TV
  - Laptop
  - Lights
- Kitchen's smart devices
  - Lights

There are also some participants:

- Persons
  - Bob

    - Alice
    - Guest 1
- Pet
    - Goldfish

We have implement four agents, one SA per room and one MA, to whom the SAs are attached.

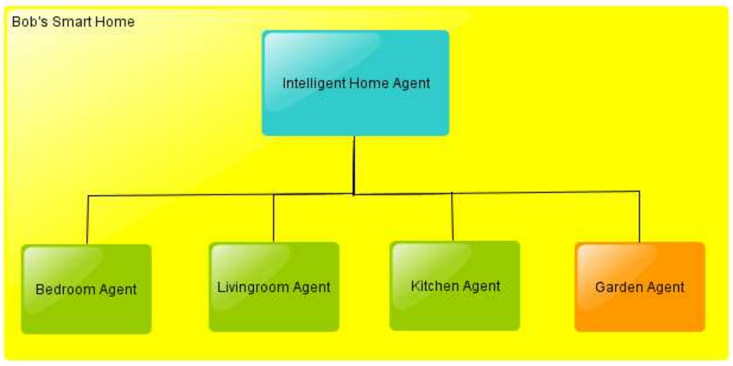


Figure 4-2: Smart Home's Agents' interconnection

4.2 Scenarios

Five scenarios have been implemented in order to demonstrate the basic reasoning tasks of our system. Each scenario is representational for a group of cases, which are respectively:

- Simple scenarios
- Scenarios with duration
- Scenarios with HCDs (Hidden Casual Dependencies)
- Scenarios with rules' priorities and preferences
- Scenarios with contradictious facts and conflicts between the different agents knowledge bases (proof of evidence)

4.2.1 Scenario 1: Turn on/off the lights

Each room agent must turn on the room's lights when a person is in the room and turn them off when there is nobody in.

The agent in the living room (LA) can sense:

- Whether a person is in the room

- Whether a person cross a door and moves
  - From living room to another room
  - From the other rooms back to the living room
  - Or outside the house

The other room agents can’s sense the presence of a person in the room but the living room agent can inform them whether a person is in their range or not, through the MA.

This group of cases includes simple scenarios, where knowledge is distributed through agents without conflicting conclusions. Each agent adapts the new pieces of knowledge and evaluates its new state.

The three room agents of living room (LA), bedroom (BA) and kitchen (KA) are attached to the master agent IHA. The LA senses the fluent ‘PersonLocation(Person,Room)’. IHA is informed that this fluent is sensed by LA. When Bob is in the living room, PersonLocation(Bob,LivingRoom) holds. LA senses and distributes that piece of knowledge to master agent IHA. The other two room agents have informed IHA that the fluent PersonLocation is missing information for them. Whenever the fluent’s state is changed, IHA informs these two agents about the new state. The agents adopt the new state. LA can sense the event ‘PersonMoveTo(Person,Room1,Room2)’, which represents the event that a person has just moved from Room1 to Room2. The event can alter the state of the fluent PersonLocation. If there were held PersonLocation(Bob,LivingRoom) and the event PersonMoveTo(Person,LivingRoom,Bedroom) is sensed, then the previous state is removed and now holds PersonLocation(Bob, Bedroom). LA will distribute the new state to IHA and IHA will inform the BA and KA. At that time BA will reason that a person is in the room at it will turn on the room’s lights. If Alice is also detected to enter in the bedroom, nothing will happen as the lights are already on. When all persons that have entered the room, return to living room, if the lights are still on, BA will turn them off.

4.2.2 Scenario 2: The Glassfish

Bob has a glassfish in his bedroom. The glassfish has to be feed twice per day. From one feeding to the other, there must be at least ten hours and at most twelve hours elapsed time. The glassfish’s feed is in the kitchen and the kitchen agent can sense when somebody is picking up the feed. If the ten hours from the last feeding hasn’t elapsed when someone picks up the feed, then a light notification in the kitchen is activated for five seconds. If there has been twelve hours from the last feeding, then a light notification in the living room is activated. The bedroom agent senses when the glassfish is being fed and distributes suitable pieces of information to the MA.

A similar scenario can be implemented for the attendance of a person's medication.

This group of cases includes scenarios where we want to reason in real time about events that take part in logic conditions like "$t_1 < t_2$" or "$n < m$" and the "n^{th}" occurrence. Specifically, in the first occasion, we want to find out when the logic condition is violated or not along the spaces before t_1, between t_1 and t_2 and after t_2. This kind of cases is common in AMI and business applications.

In the glassfish scenario there are three logic conditions: 1. the time space between the first fed and ten hours later, 2. the time space between ten and twelve hours later and 3. the time space twelve hours later. In our scenario we can conclude that the condition starts when the glassfish is fed in the first time (the space before t_1). We can reason that the condition between t_1 and t_2 is violated or not when we try to feed the fish before the ten hours time elapsed and in the space between ten and twelve hours elapsed. Finally, we can reason that the condition after t_2 is violated or not when the twelve hours have elapsed.

We also can reason about the "n^{th}" fed of the glassfish. In particular, we can reason about the 2nd fed of the glassfish in a day.

4.2.3 Scenario 1.2: Electric power – lights HCD

We extend scenario 1 by adding a constraint which indicates that in order a light device to be turned on, there must be electric power in the room. If we know that a person is in a room at t=0 but we have no knowledge about the electric power we have no clue about the room's lights. But if in a future time point we learn that in t=0 the room has electric power, and no changes have been made so far, we can conclude that the lights from time point 0 until now, remain open.

This group of cases includes scenarios where there are missing pieces of knowledge which may become know in a future time. The scenario just makes use of the DECKT reasoning process, combined with the simple case of distributed reasoning among agents.

The fluent 'Open(Light)' represents the knowledge that Light is turned on and the fluent 'HasElectricPower(Room)' represents the knowledge that the Room has electric power. When a room agent senses an event for turning on a light at t=0, if the agent has no knowledge about the room's electric power, the agent will reason that it should hold the HCD

KB:

$$(Open(Light) \vee (\neg HasElectricPower(Room)))$$

If at t=2 the agent is informed that at t=0 the room has electric power then it will reason that $Open(Light)$ at t=0

KB:

$$Open(Light)$$

$$HasElectricPower(Room)$$

If no changes had occurred in the meantime, the agent will also reason that $Open(Light)$ at t=1 and t=2. The HCD is destroyed.

If the agent had sensed at t=1 that the light was turned off, it would had reasoned that

KB:

$$(\neg Open(Light))$$

regardless of the state of electric power and the HCD would had been destroyed. The new piece of information at t=2 would not had changed that current state.

In all other cases where knowledge about the room's electric power is granted, the reasoning process is according to the scenario's description.

4.2.4 Scenario 4: Video Call

Bob is receiving a video call and we want this call to be served by some device. The device must be in same room as Bob. If more than one device exists in the room that can serve a video call, we can choose the most preferable one. If no such device exists in the room, we choose the default device that serves video calls (a smart phone). The initiation of a video call is sensed by the home's MA. The selection of the most suitable device is also made by the MA after consulting the knowledge that the SAs have distributed to it.

This group of cases includes scenarios where we want to express priority or preference on specific possibly worlds. According to our scenario, if there can be selected only one device to serve the video call, then the number of possible worlds is equivalent to the number of those devices that are suitable to serve the call. Although these worlds are equally likely, we arbitrarily choose one of them as the true state of our environment and reject all the others. The selection is made through our preferences or priorities among the possible worlds. We could also extend this group of cases in order to express probability among worlds and choose the more possible one.

4.2.5 Scenario 5: Bob’s location

We extend the basic implementation of the smart home by adding one more SA, the garden SA, which senses a person’s presence just like the living room. What will happen if both agents result that Bob is recognized in their region. As Bob can’t be in two rooms simultaneously, the agents that control those devices must communicate in order to resolve the affair. The involved agents will have to pass through a proof of evidence mechanism.

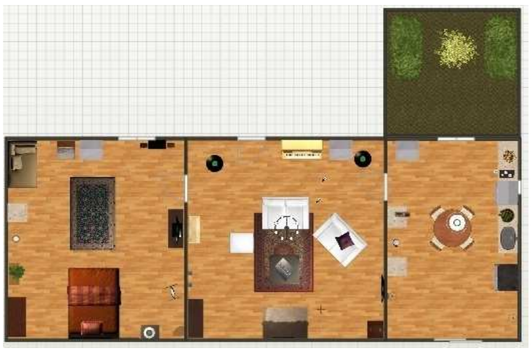


Figure 4-3: The Smart Home

This group of cases includes scenarios with contradictious facts and conflicts between the different agents’ knowledge bases. The proof of evidence mechanism is introduced in order to resolve this kind of affairs. When contradictious facts and conflicts arise, more than one possible world is encountered. The proof of evidence mechanism is used in order to select one possible world and keep our system consistent and correct.

If Bob moves from living room to kitchen, the living room agent will sense that event. Then Bob moves from kitchen to garden, which will be noticed by the garden agent. Suppose that the living room agent would possess a high degree of certainty (9) for the first movement but the garden agent possesses a low degree of certainty (4) for the second movement. If the proof of evidence mechanism had used the certainty degree feature to resolve the conflict, it would falsely stop at step 4, resulting that Bob is still in the kitchen. Only the cases where the proof of evidence mechanism proceeds in the share theory creation, would resolve the affair correctly.

Suppose that Bob is in the living room is in the living room at 12:00:00. Then he moves to kitchen at 12:05:00. The event is sensed by LA (PersonMoveTo(Bob,Livingroom,Kitchen)). Bob goes outside to the garden at 12:10:00. The event is sensed by GA (PersonMoveTo(Bob,Kitchen,Garden)). If the MA constructs a share theory, the following event history will be determined:

- HoldsAt(PersonLocation(Bob,Livingroom),0)
- Happens(PersonMoveTo(Bob,Livingroom,Kitchen),1)

- Happens(PersonMoveTo(Bob,Kitchen,Garden),2)

The fluent *'HoldsAt(PersonLocation(Bob,Livingroom),0)'* denotes that Bob is in the living room at t=0. The event *'Happens(PersonMoveTo(Bob,Livingroom,Kitchen),1)'* denotes that Bob is in the living room at the moment (t=1) and now he is moving to the kitchen. The event PersonMoveTo influences the fluent PersonLoction. Then it holds *'HoldsAt(PersonLocation(Bob,Livingroom),2)'*. Both agents possess the same rules for the manipulation of PersonLocation and PersonMoveTo.

Rule 1 & 2: The influence of PersonMoveTo in PersonLocation

$$[person, room1, room2, time]$$

$$Happens(PersonMeveTo(person, room1, room2), time)\ \&$$

$$HoldsAt(PersonLocation(person, room1), time)\ \rightarrow$$

$$Terminates((PersonMeveTo(person, room1, room2), PersonLocation(person, room1), time).$$

$$[person, room1, room2, time]$$

$$Happens(PersonMeveTo(person, room1, room2), time)\ \&$$

$$HoldsAt(PersonLocation(person, room1), time)\ \rightarrow$$

$$Initiates((PersonMeveTo(person, room1, room2), PersonLocation(person, room2), time).$$

Rule 3: A person can be only in one room at a time

$$[person, room1, room2, time]$$

$$HoldsAt(PersonLocation(person, room1), time)\ \&$$

$$HoldsAt(PersonLocation(person, room2), time)\ \rightarrow$$

$$room1 = room2.$$

The above three rules will compose the share theory. The master agent will perform a reasoning process for this share theory and the events' history. The final model is calculated and the fluent's state is determined:

- ***Final Model:*** PersonLocation(Bob,Garden)

The piece of knowledge is published and the agents are informed about the affair's termination.

4.3 The muddy children scenario

4.3.1 Description

In this section we are presenting the appliance of the system to a more general class of problems. We implement the ***'muddy children'*** scenario, a well-known theoretical problem for commonsense reasoning and multi-agent coordination. We highlight the generality of our approach and the ability to solve problems beyond the applications for AMI.

"Several children are playing together outside. After playing they come inside and their mother says to them, *at least one of you has mud on your head*. Each child can see the mud on others but cannot see his own forehead. She then asks the following question over and over:

Can you tell for sure whether or not you have mud on your head?

Assuming that all of the children are intelligent, honest, they never make logical mistakes nor fail to deduce something which is logically deducible, and answer simultaneously, what will happen?"

To get a feeling for what is being asked, we now figure out what happens if there are two children. First, suppose that exactly one is muddy. When the mother asks the question, the muddy child sees no mud on the other child, and then can conclude that he has mud on his forehead. The other child cannot tell whether or not he has mud on her forehead. Now, suppose that both children have mud on their forehead. When the mother asks the question, neither can determine if they have mud on their foreheads since they see the other child with mud. So, neither can answers yes to the question. Now, when the mother asks the question the second time, both children realize that they must have mud on their head; if either didn't have mud on their head, then the other child would have seen this and would have been able to answer yes the first time the mother asked the question. So, both answers yes to the second time the question is asked.

In our example, there participate three children. Then three cases are possible:

- ***Case 1:*** There is exactly *one child* with mud on his forehead.
    - After the mother asks the question once, the muddy child is able to answer yes and the other two children cannot answer yes.
- ***Case 2:*** There are exactly *two children* with mud on their foreheads.
    - After the mother asks the question once, no child is able to answer yes.
    - After the mother asks the question a second time, the children with mud on their foreheads can answer yes.

- ***Case 3:*** *All three children* have mud on their foreheads.
    - No children can answer yes after the mother asks the question for the first and second time.
    - All three children can answer yes after the third time.

It is proved that if there are ‘k’ muddy children, after the kth time the mother asks the question, the muddy children can answer yes, but they cannot answer yes before the kth time the question is asked.

4.3.2 Evaluation

For the scenario’s implementation, we use four agents:

- Three SAs for each child A, B, C of names *CAA*, *CBA*, *CCA* respectively
- One MA for the mother with name *MOA*

The children agents are connected to mother agent. Each agent can sense whether or not another person has muddy on his forehand. This piece of information is maintained as locally protected knowledge that is not distributed to the other agents. We use the fluent ‘MuddyChild’ to represent this knowledge. For example, if child B is muddy, the other agents will know that ‘MuddyChild(B)’.

The mother agent maintains the fluent ‘Questions(n)’. Initially ‘n’ is zero. Every time that the mother wants to ask the children for their state, the ‘n’ is increased by one, the fluent Question is updated and the children agents are informed about the change. The mother agent will ask as many questions as the number of the muddy children.

At first, the children agents will reason if they are muddy or not. Each child checks if the question number is larger from the number of the muddy children that this child has sensed. If this is the case, the child deduces that he is muddy. For example, in ***‘case 2’***, when the mother asks the question for the second time, the two muddy children will realize that they have sensed only one muddy child (while the third one is clean) but their mother has just informed them that the muddy children are at least two. As the states of the other children are known, these children deduce that they are muddy. A muddy child uses the fluent ‘IKnowIAmMuddy’ to declare this kind of knowledge and distributes it to the other agents. If children B and C are the muddy ones, they will distribute the fluent IKnowIAmMuddy(B) and IKnowIAmMuddy(C) respectively.

On the other hand the clean children can reason about their state when the muddy ones declare that they know their state. The clean children can sense the muddy ones and their mother’s questions. When all the muddy children, that have been previously sensed, are denoting that they know that they are muddy and the question number is equal to their number, each one of the clean children can reason that they came to their conclusions by taking into consideration its state. So, the clean children deduce that they are clean. The fluent ‘IKnowIAmClean’ is used to declare this kind of knowledge. In our case, the child A will distribute the fluent IKnowIAmClean(A).

5. Conclusions

To conclude this thesis, we summarize and discuss its main contributions and propose possible directions for future research.

5.1 Synopsis

The imperfect nature of context knowledge and the special characteristics of Ambient Intelligence environments have introduced new challenges in the field of Distributed Artificial Intelligence. The knowledge distribution is one of these open issues. An agent is likely to require pieces of information about the environment that are not accessibly by him. Thus, agents must join in a network where each agent would be able to search for those missing pieces of information and get informed about future changes that could occur. The global consistency of the whole system is a basic field of research for this study. As agents are sensing the environment variables, incorrect information can arise from missing facts and ambiguous information between the different agents' perceptions. We model agents as nodes in a peer-to-peer network, considering the conflicts that may arise during the integration of the knowledge distribution.

Two main features are proposed by this thesis: the *reasoning process* and the *proof of evidence mechanism*. The reasoning process is based on a data type that takes use of Java Timer objects and enables agents to reason about real system time. Thus, cases like the conditional 'n>m' or the 'n^{th} occurrence' can be modeled. Priorities and preferences can also be expressed and modeled in the reasoning process by exploiting the capabilities of Jess. Epistemic reasoning is also feasible, due to DECKT, making the system more robust in coexistence with the imperfect nature of the available context information.

The proof of evidence mechanism is used to resolve the affairs that could arise during the integration of the distributed knowledge. Composed by two features, the certainty degree and the share theories, this mechanism can resolve several cases of conflicts. The grading mechanism of the certainty degree is used for the fast resolution of an affair when locally protected knowledge about security or privacy data is used; making use of subjective norms to determine which possible world is the most likely. The share theories are the main feature for conflict resolution. Occasions where problems can occur for both features are noticing. In most cases the system will be globally consistent. However, when problems occur, it will always remain at least in coherent state and it will never be inconsistent.

5.2 Future Directions

This thesis deals with the distributed reasoning process about knowledge and epistemic knowledge. The basic reasoning process could be further evolved to support reasoning about belief, probability and their combinations with knowledge. Planning administration in such a multi-agent system could also be examined in combination with the proposed reasoning process and the proof of evidence mechanism. So far, our approach assumes that agents are always willing to disclose information available to them, except if explicitly reported not to, in cases of locally protected knowledge. The integration of agents’ goals and desires, as game theories between the agents should also be part of research. Coherent we the above issues, access policies for the distributed knowledge among the agents or group of agents, security and privacy issues may be also taken under consideration. Finally, additional scenarios and case studies, which will further reveal the capabilities and the disabilities of the proposed system, might be implemented.

6. Bibliography

1. Antonis Bikakis, Grigoris Antoniou: *Distributed Reasoning with Conflicts in an Ambient Peer-to-Peer Setting.* In Constructing Ambient Intelligence. Communications in Computer and Information Science, 2008, Volume 11, Part 1, 24-33. (2008)
2. F. Rossi, P. van Beek, T. Walsh: *Handbook of Constraint Programming*. Elsevier, Foundations of Artificial Intelligence, Series Editors: I.Hendler, H. Kitano, B. Nebel. Chapter 20: Distributed Constraint Programming. Page 700. (2006)
3. Armando Roy Delgado, Rubén Blasco, Álvaro Marco, Diego Cirujano, Roberto Casas, Armando Roy Yarza, Vic Grout, Richard Picking: *Agent-Based AmI System Case Study: The Easy Line + Project.* Centre for Applied Internet Research. Paper 48. (2010)
4. Erik T. Mueller: *Commonsense Reasoning*. Elsevier, Morgan Kaufman Publishers. (2006)
5. Erik T. Mueller: *Discrete Event Calculus Reasoner Documentation*. IBM Thomas J. Watson Research Center. (2008)
6. Murray Shanahan: *The Event Calculus Explained.* In Artificial intelligence today. Lecture Notes in Computer Science, 1999, Volume 1600/1999, 409-430 (1999)
7. Theodore Patkos: *A Formal Theory for Reasoning about Action, Knowledge and Time*. Institute of Computer Science Foundation for Research and Technology Hellas. (2010)
8. Theodore Patkos, Dimitris Plexousakis: *Reasoning with Knowledge, Action and Time in Dynamic and Uncertain Domains.* In Proceeding of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09). (2009)
9. Theodore Patkos, Dimitris Plexousakis: *Epistemic Reasoning for Ambient Intelligence.* Institute of Computer Science Foundation for Research and Technology Hellas. (2011)
10. Ron van der Meyden: *On Notions of Causality and Distributed Knowledge.* In Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning. (2008)
11. Jelle Gerbrandy: *Distributed Knowledge.* ILLC/Department of Philosophy; University of Amsterdam. (1998)
12. Floris Roelofsen: *Distributed Knowledge.* In Journal of Applied Non-Classical Logics, Volume 12 – No. -/2006. (2007)
13. Yoav Shoham, Kevin Leyton-Brown: *Multiagent Systems. Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press. (2009)
14. Verónica Venturini, Javier Cardó, José Manuel Molina: *An Ambient Intelligent Platform based on Multi-Agent System.* In 11th Workshop of Physical Agents 2010. (2010)

15. Michael Wooldridge, Nicolas R. Jennings, David Kinny: *The Gaia methodology for Agent-Oriented Analysis and Design.* In Autonomous Agents and Multi-Agent Systems. Volume 3, Number 3, 285-312. (2000)
16. Pavlos Moraitis, Nikolaos I. Spanoudakis: *Combining Gaia and JADE for Multi-Agent Systems Development.* In Proceedings of the 17th European Meeting on Cybernetics and Systems Research (EMCSR 2004), Vienna, Austria, April 2004. (2004)
17. Ioannis L. Tambakis: *Automating Commonsense Reasoning: A Satisfiablility-Based Commonsense Reasoning System Incorporating a Tool for Measuring the SAT Progress.* Institute of Computer Science Foundation for Research and Technology Hellas. (2008)
18. Papatheodorou Constantinos: *A Distributed Defeasible Reasoning System for Mobile Devices in Ambient Intelligence Environments.* Institute of Computer Science Foundation for Research and Technology Hellas. (2010)
19. Manuel Garcia-Herranz del Olmo: *Easing the Smart Home: a rule-based language and multi-agent structure for end user development in Intelligent Environments.* In Journal of Ambient Intelligence and Smart Environments. IOS Press, Volume 2, November 4 / 2010, pages 437-438. (2010)
20. Jose Viterbo, Laurent Mazuel, Yasmine Charif, Markus Endler, Nicolas Sabouret, Karin Breitman, Amal El Fallah Seghrouchni, Jean-Pierre Briot: *Ambient Intelligence: Management of Distributed and Heterogeneous Context Knowledge.* CRC studies in Informatics Series, Chapman & Hall, pp. 1-44. (2008)
21. Dante I. Tapia, Ajith Abraham, Juan M. Corchado, Ricardo S. Alonso: *Agents and ambient intelligence: case studies.* In Journal of Ambient Intelligence and Humanized Computing. Volume 1, Number 2, 85-93. (2010)
22. Antonis Bikakis, Theodore Patkos, Grigoris Antoniou, Dimitris Plexousakis: *A Survey of Semantics-based Approaches for Context Reasoning in Ambient Intelligence.* In Constructing Ambient Intelligence. Volume 11, Part 1, 14-23. (2008)
23. Matteo Bonifacio, Paolo Bouquet, Gianluca Mameli, Michele Nori: *Peer-Mediated Distributed Knowledge Management.* In Agent-Mediated Knowledge Management. Lecture Notes in Computer Science, 2004, Volume 2926/2004. (2004)
24. Bala M. Balachandran: *Developing Intelligent Agent Applications with JADE and JESS.* In Knowledge-Based Intelligent Information and Engineering Systems. Lecture Notes in Computer Science, 2008, Volume 5179/2008. (2008)
25. Joel Vogt: *Jess to JADE Toolkit (J2J) – A Rule-based Solution Supporting Intelligent and Adaptive Agents.* Software Engineering Group Department of Informatics, University of Fribourg (Switzerland). (2008)
26. Wolfgang Laun: *Predicate Calculus and Jess.* Thales Rail Signaling GesmbH Vienna, Austria. (2009)

27. Joan Ametller, Sergi Robles, Joan Borrell: *Agent Migration over FIPA ACL Messages.* In Mobile Agents for Telecommunication Applications. Lecture Note in Computer Science, 2003, Volume 2881/2003, 210-219. (2003)
28. Solomon Eyal Shimony, Ephraim Nissan: *Kappa calculus and evidential strength: A note on Aqvist's logical theory of legal evidence.* In Artificial Intelligence and Law. Volume 9, Number 2-3, 153-163. (2001)
29. Marco Montali, Fabrizio M. Maggi, Federico Chesani, Paola Mello, Wil M.P. van der Aalst: *Monitoring Business Constraints with the Event Calculus.* DEIS Technical Report no. DEIS-LIA-002-11. LIA Series no. 100. (2011)
30. Kevin I-Kai Wang: *Multi-agent based Ambient Intelligence Platform.* Electrical and Electronic Engineering, University of Auckland. (2009)
31. Andrei Olaru: *A Context-Aware Multi-Agent System for AmI Environments.* Technical report, University Politehnica of Bucharest, University Pierre et Maria Curie Paris. (2010)
32. Theodore Patkos, Ioannis Chrysakis, Antonis Bikakis, Dimitris Plexousakis, Grigoris Antoniou: *A Reasoning Framework for Ambient Intelligence.* In Artificial Intelligence: Theories, Models and Applications. Lecture Note in Computer Science, 2010, Volume 6040/2010, 213-222. (2010)
33. Theodore Patkos, Antonis Bikakis, Grigoris Antoniou, Maria Papadopouli, Dimitris Plexousakis: *Distributed AI for Ambient Intelligence: Issues and Approaches.* In AmI'07 Proceedings of the European conference on Ambient Intelligence. (2007)
34. Grigoris Antoniou, Constantinos Papatheodorou, Antonis Bikakis: *Reasoning about Context in Ambient Intelligence Environments: A Report from the Field.* In 12th International Conference on the Principles of Knowledge Representation and Reasoning. (2010)
35. Antonis Bikakis, Grigoris Antoniou: *Distributed Defeasible Contextual Reasoning in Ambient Computing.* In Ambient Intelligence. Lecture Notes in Computer Science, 2008, Volume 5355/2008, 308-325. (2008)
36. Antonis Bikakis, Theodore Patkos, Grigoris Antoniou, Dimitris Plexousakis: *A Survey on the Use of Semantics-Based Technologies for Ambient Intelligence.* In Constructing Ambient Intelligence. Communications in Computer Science and Information Science, 2008, Volume 11, Part 1, 14-23. (2008)
37. Murray Shanahan: *The Ramification Problem in the Event Calculus.* In International Joint Conference on Artificial Intelligence. (1999)
38. Murray Shanahan: *Event Calculus Planning Revisited.* In Recent Advances in AI Planning. Lecture Notes in Computer Science, 1997, Volume 1348/1997, 390-402. (1997)
39. *Running JADE under Eclipse:*http://wrjih.wordpress.com/2008/11/29/running-jade-under-eclipse/?replytocom=9054#respond
40. *DECKT:* http://www.csd.uoc.gr/~patkos/deckt.htm
41. *Jess:* http://www.jessrules.com/jess/
42. *agentTool III:* http://agenttool.cis.ksu.edu/index.php
43. *O-MaSE:* http://macr.cis.ksu.edu/home.html

44. *Integrating JADE and Jess:* http://jade.tilab.com/doc/tutorials/jade-jess/jade_jess.html
45. *(Wikipedia) Intelligent agent:* http://en.wikipedia.org/wiki/Intelligent_agent
46. *(Wikipedia) Software agent:* http://en.wikipedia.org/wiki/Software_agent
47. *(Wikipedia) Multi-agent system:* http://en.wikipedia.org/wiki/Multi-agent_system
48. *(Wikipedia) AmI:* http://en.wikipedia.org/wiki/Ambient_intelligence
49. *(Wikipedia) Closed World Assumption:* http://en.wikipedia.org/wiki/Closed_world_assumption
50. *(Wikipedia) Negotiation as failure:* http://en.wikipedia.org/wiki/Negation_as_failure
51. *(Wikipedia) Circumscription:* http://en.wikipedia.org/wiki/Circumscription_(logic)
52. *Epistemic Logic:* http://plato.stanford.edu/entries/logic-epistemic
53. *(Wikipedia) Consistent Theory:* http://en.wikipedia.org/wiki/Consistency
54. *(Wikipedia) Coherence Theory:* http://en.wikipedia.org/wiki/Coherentism
55. Massimiliano Garagnani, Maria Fox, Derek P. long: *Belief Systems for Conflict Resolution.* In Tessier and Chaudron 1998, 55-60. (1998)
56. Céline Darnon, Céline Buchs, Fabrizio Butera: *Epistemic and relational conflicts in sharing identical vs. complementary information during cooperative learning* In Swiss Journal of Psychology. Publisher Verlag Hans Huber, Volume 61, Number 3 / 2002, pages 138-151. (2002)