ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΑΝΑΠΤΥΞΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΓΛΩΣΣΑΣ ΓΙΑ ΤΗΝ ΔΙΑΔΙΚΑΣΙΑ ΑΔΕΙΟΔΟΤΗΤΗΣ ΣΕ ΠΕΡΙΒΑΛΛΟΝΤΑ ΔΙΑΧΥΤΗΣ ΝΟΗΜΟΣΥΝΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Γενιτσαρίδη Ειρήνη

Ηράκλειο

Ιούνιος 2011

An Authorization Language in Ambient Intelligence Environments

Genitsaridi Eirini

Master of Science Thesis

Computer Science Department, University of Crete

Abstract

Ambient Intelligence (AmI) is a new wave of information technology that integrates microprocessors into everyday objects in order to improve the quality of everyday life. The information is distributed among various devices that collect, process, change and share it.

As a new paradigm of information technology, Ambient Intelligence has introduced new research challenges in many areas including the field of authorization. The implementation of authorization policies is vital in order to develop a secure AmI system. Every Ambient Intelligence device should be able to specify access rights policies to the resources that it controls. However, the distributed and often imperfect information, the open and dynamic nature of AmI environments and the special characteristics of the involved devices make the enforcement of authorization policies problematic.

Previous work by Bikakis et. al. presented *Contextual Defeasible Logic (CDL*), a fully distributed approach for reasoning with conflicts in Ambient Intelligence systems. Here we extend this approach to address authorization issues in distributed environments. We present *Distributed Environment Authorization Logic* (*DEAL),* a formal high level logic-based language to specify access control policies in open and dynamic distributed systems. The language has rich expressive power supporting negative authorizations, rule priorities, hierarchical category authorizations and nonmonotonic reasoning. We define the language semantics through Defeasible Logic. We also demonstrate DEAL authorization policies in two concrete implemented Ambient Intelligence scenarios.

Ανάπτυξη και Υλοποίηση Γλώσσας για την Διαδικασία Αδειοδότησης σε Περιβάλλοντα Διάχυτης Νοημοσύνης

Γενιτσαρίδη Ειρήνη

Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών

Περίληψη

Η Διάχυτη Νοημοσύνη είναι ένα νέο κύμα τεχνολογίας πληροφοριών που ενσωματώνει μικροεπεξεργαστές σε καθημερινά αντικείμενα προκειμένου να βελτιωθεί η ποιότητα της καθημερινής ζωής. Οι πληροφορίες είναι κατανεμημένες μεταξύ διάφορων συσκευών που τις συλλέγουν, επεξεργάζονται, μεταβάλλουν και μοιράζονται. Ως νέο παράδειγμα τεχνολογίας πληροφοριών, η Διάχυτη Νοημοσύνη έχει δημιουργήσει νέες προκλήσεις σε πολλές ερευνητικές περιοχές συμπεριλαμβανομένου του τομέα της αδειοδότησης. Η εφαρμογή των πολιτικών αδειοδότησης είναι ζωτικής σημασίας για την ανάπτυξη ενός ασφαλούς συστήματος Διάχυτης Νοημοσύνης. Κάθε συσκευή Διάχυτης Νοημοσύνης πρέπει να είναι σε θέση να προσδιορίσει πολιτικές δικαιωμάτων πρόσβασης στους πόρους που ελέγχει. Εντούτοις, οι κατανεμημένες και συνήθως ελλιπείς πληροφορίες, η ανοικτή και δυναμική φύση των περιβαλλόντων Διάχυτης Νοημοσύνης και τα ειδικά χαρακτηριστικά των εμπλεκόμενων συσκευών δημιουργούν προβλήματα στην επιβολή των πολιτικών αδειοδότησης.

Προηγούμενη έρευνα (Bikakis et. al.) παρουσίασε την Αναιρέσιμη Συλλογιστική Περιβάλλοντος (Contextual Defeasible Logic ή CDL), μια πλήρως κατανεμημένη προσέγγιση για συλλογιστική με συγκρούσεις σε περιβάλλοντα Διάχυτης Νοημοσύνης. Εδώ επεκτείνουμε αυτήν την προσέγγιση ώστε να χειρίζεται ζητήματα αδειοδότησης σε κατανεμημένα περιβάλλοντα. Παρουσιάζουμε την Γλώσσα Αδειοδότησης Κατανεμημένου Περιβάλλοντος (Distributed Environment Authorization Logic ή DEAL), μια επίσημη υψηλού επιπέδου λογική γλώσσα για να προσδιορίζουμε πολιτικές πρόσβασης πόρων σε ανοικτά και δυναμικά κατανεμημένα

συστήματα. Η γλώσσα έχει πλούσια εκφραστική δύναμη υποστηρίζοντας αρνητικές άδειες, προτίμηση σε αντικρουόμενους κανόνες, άδειες σε ιεραρχημένες κατηγορίες και μη μονοτονικό συλλογισμό. Ορίζουμε τη σημασιολογία της γλώσσας μέσω της Αναιρέσιμης Συλλογιστικής. Περιγράφουμε επίσης την εφαρμογή πολιτικών αδειοδότησης σε δύο συγκεκριμένα υλοποιημένα σενάρια Διάχυτης Νοημοσύνης.

Επόπτης Καθηγητής:

Γρηγόρης Αντωνίου

Καθηγητής Τμήματος Επιστήμης Υπολογιστών

Πανεπιστημίου Κρήτης

List of Figures

Figure 1.1: Related areas to Ambient Intelligence 5

Figure 1.2: Context information flow in the AmI hospital scenario.....8

Figure 1.3: Context information flow in the AmI university scenario 12

Figure 2.1: The request-pair of a simple Ambient Intelligence example. 17

Figure 2.2: An authorization example in language [29]. 18

Figure 2.3: Two query examples of service requests.....18

Figure 2.4: An example of conflicting rules in language of [29]. 22

Figure 3.1: An authorization policy with negative authorizations.....25

Figure 3.2: Context information flow in the scenario.....26

Figure 3.3: An authorization policy that requires rule priorities. 27

Figure 3.4: An example of user hierarchical categories. 28

Figure 3.6: An example of action hierarchical categories. 29

Figure 3.7: An example of object hierarchical categories. 29

Figure 3.8: An authorization policy with nonmonotonic reasoning. 31

Figure 4.1: Model of authorization in Woo, Lam approach [45]. 37

List of Tables

Table of Contents

List of Figures ..... **vi**

List of Tables ..... **vii**

1. Introduction **4**

1.1 Ambient Intelligence 4

1.2 Authorization in Ambient Intelligence 5

1.3 Motivating Scenarios 7

1.3.1 Ambient Intelligence Hospital Scenario 7

1.3.2 Ambient Intelligence University Scenario 10

1.4 Approach 14

1.5 Thesis Contribution 15

1.6 Thesis Organization 15

2. Basic Concepts of the Authorization Problem **17**

2.1 Request-Pair 17

2.2 Authorization 17

2.2.1 Service 18

2.2.2 Grantor 19

2.2.3 Grantee 19

2.3 Authorization Conflict 19

2.4 Authorization Policy 21

3. Desirable Characteristics of an Authorization Language **24**

3.1 Negative Authorization 24

3.2 Rule Priorities 25

3.3 Hierarchical Category Authorization 28

3.4 Nonmonotonic Reasoning 30

4. Related Work 32

4.1 Non logic-based Authorization Approaches 35

4.2 Logic-based Authorization Approaches 37

4.2.1 Centralized Authorization Approaches 38

4.2.2 Decentralized Authorization Approaches 39

5. Background Information 40

5.1 Defeasible Logic 40

5.1.1 Proof Theory 42

5.2 Multi-Context Systems 43

5.3 Contextual Defeasible Logic 45

5.3.1 Representation Model 45

6. A Distributed Environment Authorization Language: DEAL 47

6.1 Language Syntax 47

6.1.1 Alphabet of DEAL Language 47

6.1.2 Rules of DEAL Language 50

6.1.3 Characteristics of DEAL Language 53

6.2 Language Semantics 57

6.2.1 DEAL alphabet transformation 57

6.2.2 DEAL rules transformation 58

6.3 Contextual Defeasible Logic Extensions 61

6.4 Motivating Scenarios Implementation 71

6.4.1 Implementation of AmI Hospital Authorization Scenario 71

6.4.2 Implementation of AmI University Authorization Scenario 75

7. Conclusion 79

7.1 Synopsis 79

7.2 Future Directions 80

8. Appendix A 82

A.1 TuProlog Reasoner 82

A.2 Defeasible Logic Metaprogram 82

A.3 DEAL Metaprogram 84

9. Bibliography 86

Introduction

Access control is the ability of a system to prohibit unauthorized entities to consume specific system services. In physical security, the term access control refers to the practice of restricting entrance to physical objects such as a property, a building or a room to authorized persons (e.g. ticket inspector in a bus). In computer security, access control refers to any mechanism that manages the admission to computer services such as accessing system information or performing some action to system resources (e.g. update information in a Web server).

Access control is a very important topic in the development of nowadays computer applications. Companies usually require access control in order to grant access to areas and information only to individual users and groups with the appropriate permission level. Access control is crucial in systems that include sensitive data such as medical information in hospital facilities, political beliefs in online voting systems, bank account passwords in e-commerce systems or religion and sex preferences in social networks.

Access control involves various measures such as biometric scans and metal locks, digital signatures, encryption, camera monitoring and others. Moreover access control consists of three basic processes, Authentication, Authorization and Auditing (AAA). Authentication is the process of verifying if the identity that a requester provided is authentic. Authentication answers the question: Is the requester who he claims to be? Authorization is the process that determines if a requester is permitted to consume a specific service according to various system policies. Authorization answers the question: Is the requester permitted to consume this service? Accountability is the process of maintaining a record of actions performed by every requester (successful or failed attempts to consume services). In this research we study the process of authorization in the Ambient Intelligence domain.

1.1 Ambient Intelligence

Ambient intelligence (AmI) is a new wave of information technology that typically integrates microprocessors into everyday objects in order to improve the quality of everyday life. AmI environments include heterogeneous intelligent devices that

communicate by means of ad-hoc wireless networks. Every intelligent device acts as an autonomous entity that controls resources, handles requests and sends requests to other entities. The core difference between AmI and traditional computer systems is their user centric approach. AmI systems adapt and respond to people by acknowledging their presence and gestures instead of the other way around. Therefore an Ambient Intelligence system can be seen as the most evolved form of a computer system that requires the minimal user interaction in order to adjust to the user's needs. A simple example of Ambient Intelligence is a house with the ability to acknowledge human presence in a variety of places and adjust the light accordingly.

Ambient Intelligence is a multidisciplinary approach as presented in [1, 4], since it requires the convergence of many areas in Computer Science in order to fulfill its purpose. The relevant areas are depicted in Figure 1.1.

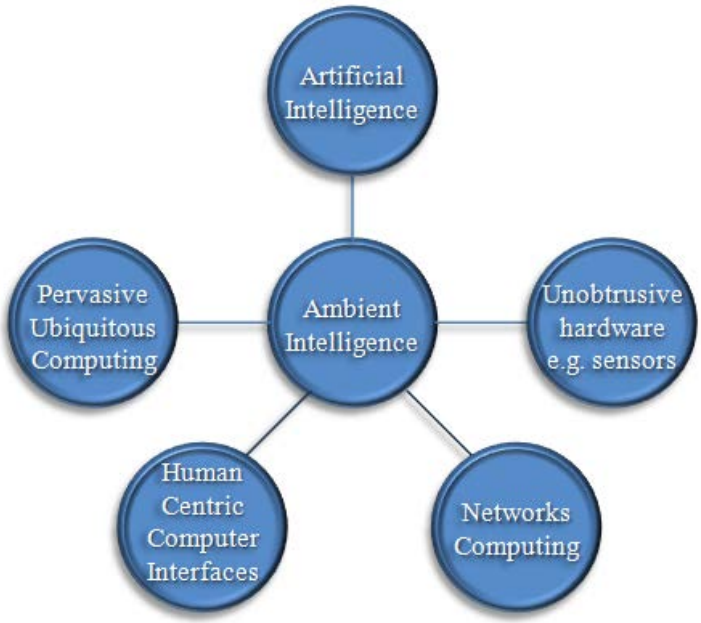

Figure 1.1: Related areas to Ambient Intelligence

1.2 Authorization in Ambient Intelligence

Ambient Intelligence systems aim at providing the right information or behavior to the right users, at the right time, in the right place. In order to achieve this, a system must have a thorough knowledge and, as one may say, "Understanding" of its environment, the people and devices that exist in it, their interests and capabilities,

and the tasks and activities that are being undertaken. All this information falls under the notion of context. Dey et al. [6] described context as "any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and application, including the user and applications themselves". Other context definitions can be found in [7-9]. An example of context information in a computer application for a hospital can be information about the role of a person as doctor, patient or nurse.

A special characteristic of Ambient Intelligence environments is the imperfect nature of context information. Henricksen and Indulska in [10] characterize four types of imperfect context information: unknown, ambiguous, imprecise, and erroneous. Sensor or connectivity failures (which are inevitable in wireless connections) result in situations, that not all context data is available at any time. When data about a context property comes from multiple sources, then context may become ambiguous. Imprecision is common in sensor-derived information, while erroneous context arises as a result of human or hardware errors.

Another special characteristic of AmI environments is their open and dynamic nature. In an open and dynamic environment participating entities enter or leave the environment regularly and cannot be predetermined. The entities that operate in an Ambient Intelligence environment are expected to have different goals, experiences and perceptive capabilities. They may use distinct vocabularies and they may even have different levels of sociality. Moreover, due the unreliable and restricted (by the range of the transmitters) wireless communications, not all entities are present at a specific time instance and direct communication with all of them may be impossible.

The special characteristics of ambient intelligence environments have introduced new research challenges in many areas, as presented in [1-5], including the field of authorization. The implementation of authorization policies is vital in order to develop a secure Ambient Intelligence system. Every AmI device should be able to specify access right policies to the resources that it controls. However, the imperfect nature of context information and the open and dynamic characteristics of AmI environments make the enforcement of authorization policies problematic.

The following questions highlight some of the implications that AmI environments create to authorization as part of a system's security.

◊ How to adjust security according to context changes?

- ◊ How to protect recourses from entities when they cannot be predetermined?
- ◊ How to adjust security that relies on other entities when they leave the environment?

1.3 Motivating Scenarios

In this section we describe two concrete application scenarios in ambient intelligence environments that demonstrate the special requirements and challenges of authorization in such environments. Both scenarios require the specification of authorization policies for accessing sensitive information. The first takes place in an Ambient Intelligence hospital environment and focuses on the protection of medical data, while the second takes place in an Ambient Intelligence university and focuses on the access control of secretarial services. In section 6.4 there is a full technical description of these two interesting types of scenarios that served as motivations for our research.

1.3.1 Ambient Intelligence Hospital Scenario

A hospital usually consists of several autonomous departments that are responsible for diagnosing and treating different diseases. The motivation for this scenario is based on the fact that a doctor may send a patient to different departments for medical tests in order to diagnose his disease. The results of the exams are distributed in the different departments. Doctors usually must visit the departments periodically to ask if the results of their patients are ready. In order to automate this process, we simulated an Ambient Intelligence hospital environment and handled the raised authorization issues.

The hospital of the scenario consists of three autonomous departments, the Cardiology, the X-ray and the Gastroenterology. The Cardiology department provides medical care and performs medical procedures to patients who have problems with their heart or circulation. The X-ray department provides a full range of diagnostic imaging services such as MRI (*Magnetic resonance imaging)* scanning. The Gastroenterology department investigates and treats gastrointestinal diseases. Every

department is equipped with an intelligent computer that acts as an autonomous entity in the environment and includes information about the performed procedures.

The hospital includes also a management office which is equipped with an intelligent device that handles information about the employees (doctors, trainees, nurses etc.) and the patients that are hospitalized there. Moreover, the office maintains a variety of information about diseases. The doctors can be informed from the office about whether there is an outbreak of a specific disease or if there are disease incidents more than a specific number in order to take precaution meters. The office is also responsible for communicating with the departments in order to be updated about the exam results of the patients.

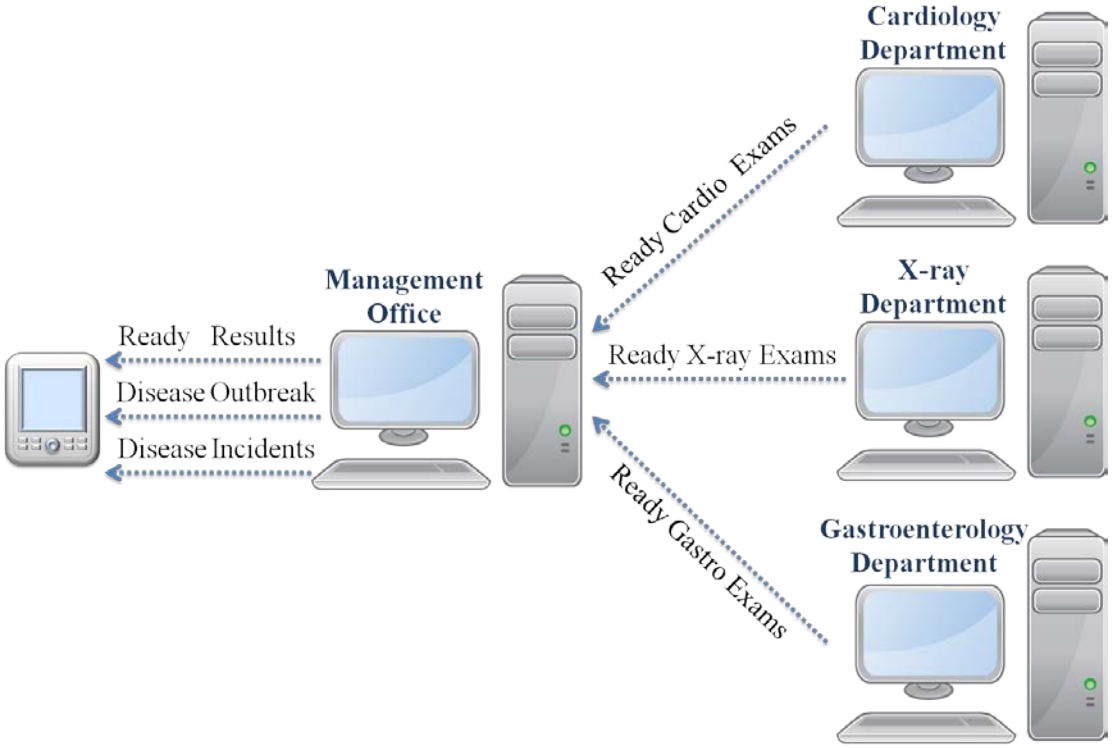

Figure 1.2: Context information flow in the AmI hospital scenario

The doctors and trainees of the hospital are provided with smart PDA computers that communicate with the management office in order to be informed about the status of the patients' exam results and the diseases information. The information flow of the scenario is depicted in figure 1.2. Essentially, the context information flow illustrates that requests about disease information (disease outbreaks

and disease incidents) are answered by the management office and requests about the status of exam results are forwarded by the management office to the appropriate department in order to be answered.

This scenario arises various authorization issues. First of all, the departments require authorization policies in order to protect their medical exams information. The authorization policy of each department is trivial, because they communicate only with the management office. Thus, departments should process requests made only by the office. Secondly, the management office should restrict its information only to requesters which are doctors that fulfill concrete attributes. The management office requires a rich and extendable authorization policy that consists of the following statements:

1. A person should be authorized to be informed about the status of a patient's exam results if he is a doctor that cures this patient.
2. A person should be authorized to be informed about the status of a patient's exam results if he is a trainee and a doctor that cures this patient permits it.
3. A person should not be authorized to be informed about the status of a patient's exam results if he is a doctor that is retired.
4. Statement 3 is preferred to statement 1.
5. A person that is a doctor or trainee belongs to the category "Doctors".
6. The persons that belong to the category "Doctors" should be authorized to be informed about a disease outbreak.
7. The persons that belong to the category "Doctors" should be authorized to be informed about disease incidents.

In order to simplify the scenario we considered a simple authorization policy for the departments. However, the scenario is easily extended to support different and more complicated authorization policies for each department. Moreover, the number of departments can be increased to fulfill the requirements of a real hospital facility.

The individuals that take part in the scenario are three doctors Bob, Trudy and Alice. The management office has the following information about these doctors:

- Bob is a doctor cures two patients Mary and George.
- Trudy is a doctor that was recently retired.

- Alice is a trainee doctor.
- Bob permits Alice to be informed about the status of George's exam results.

When Bob enters the hospital, he decides to be informed about the status of Mary's exams and tells his trainee doctor Alice to be updated about the status of George's exams. Bob uses his PDA to make a request to the management office about whether Mary's cardiology exam results are ready. The management office should authorize Bob due to statement 1 and the request should be forwarded to the Cardiology department in order to be answered. Bob also decides to make a request about if there is an outbreak of the disease 'H1N1'. The management office should conclude that Bob is authorized for this request too, due to statement 5, 6 and answer it.

Alice uses her PDA to make two requests to the management office about whether George's X-ray and Gastrointestinal exam results are ready. Alice should be granted by the management office due to statement 2 and the requests should be forwarded to the appropriate departments in order to be answered. Alice also decides to make a request about if there are more than 4 incidents of 'H1N1' in the hospital. The management office should conclude that Alice is authorized for this request too, due to statement 5, 7 and answer it.

Trudy was responsible for Bob's patient George before she was retired. Trudy decides to make a request with her PDA to the management office about whether George's X-ray exam results are ready. The management office should conclude that Trudy is not authorized for this request due to statement 1, 3 and 4.

The implementation of the above authorization statements requires an expressive language with specific characteristics that is able to support access control policies in Ambient Intelligence environments.

1.3.2 Ambient Intelligence University Scenario

A university consists of several areas where classrooms, laboratories, the office of the university secretary and administrative offices are located. These areas are used by individuals such as students, professors and university staff. The students and professors usually must visit the office of the university secretary in order to make

various requests. The motivation for this scenario is the automation of several procedures that take place in the university in order to achieve a better management of the university recourses, an easier use of the provided university services, and a decrease of the work amount in the secretary office. We simulated an Ambient Intelligence university environment where the secretary office and the individuals are equipped with intelligent computer devices that are able to communicate via wireless networks. Moreover, we handled the authorization issues that are raised from this scenario.

The university in this scenario includes a secretary office which is equipped with an intelligent device that handles information about individuals and university resources. More specifically, the computer maintains information about the students that have gotten a scholarship, the students that have successfully taken all lessons and the students that have presented the thesis. Furthermore, the device has knowledge of the persons that have completed and signed the university registration form that is required every semester. Finally, the computer maintains information of the scheduled presentations in the university classrooms.

The secretary office computer is able to provide individuals with various services. The computer provides a service that informs a student about whether he has fulfilled the requirements to get the degree (successfully taken all lessons and presented the thesis). Additionally, another service informs a student about whether he has gotten a scholarship. Moreover, the computer provides a service that informs a professor about whether a classroom is available at a specific time (there is not a scheduled presentation in the classroom at that time). Finally, another service informs the administrator about whether there is enough memory space (used memory below 80%) in the computer of the office.

The students and professors of the university are provided with smart PDA computers that communicate with the management office. The students use their PDA computers in order to be informed about scholarships and degree requirements while the professors use the devices in order to be updated about classroom reservations.

The information flow of the scenario is depicted in figure 1.3. Essentially, the context information flow illustrates that the secretary office answers requests from professors, students and administrators. In order to simplify the scenario we considered a limited set of online services that are provided by the secretary office.

However, the scenario is easily extended to support a larger number of services that fulfill the requirements of real university facilities.

Figure 1.3: Context information flow in the AmI university scenario

This scenario arises various authorization issues. The secretary office requires an expressive and extendable authorization policy in order to restrict its online services only to professors, students and administrators that fulfill concrete attributes. The authorization policy of the secretary office consists of the following statements:

1. The service that informs a student about whether the requirements to get the degree have been fulfilled belongs to the category “StudentServices”.
2. The service that informs a student about whether he will get a scholarship belongs to the category “StudentServices”.
3. A person that is a student should be authorized to consume any service that belongs to the category “StudentServices”.
4. A person that is a student should not be authorized to consume any service that belongs to the category “StudentServices” if there is not any knowledge that he has signed and completed the registration form that is required every semester.
5. Statement 4 is preferred to statement 3.
6. A person should be authorized to be informed about whether a classroom is available at a specific time, if the person is a professor of the university and there is not any knowledge that he is retired.
7. A person should be authorized to be informed about whether there is enough memory space in the computer of the office, if the person is an administrator of the university.

The individuals that take part in the scenario are professor Antoniou and three students Bob, Alice and Trudy. The secretary office has the following information about the individuals, the scheduled classroom presentations and the computer's memory status:

- Bob is a student that is registered for the current semester and has gotten a scholarship.
- Alice is a student that is registered for the current semester, has passed all the lessons of the master degree and recently presented her master thesis.
- Trudy is a student that is not yet registered for the current semester.
- Antoniou is an active professor of the university.
- The university classroom "RA201" has scheduled presentation at 5 o'clock.
- Smith is an administrator of the university computer systems.
- The computer has consumed below 80% of the memory space.

Bob decides to be informed about whether he has gotten a scholarship this semester. Bob uses his PDA to make the appropriate request to the secretary office. The secretary office should authorize Bob for this request due to statements 2, 3, 4, 5 and the information that the office maintains. Thus, the secretary office should proceed in processing the request. The information about Bob leads to a positive answer for his request. Therefore, Bob should receive a positive reply.

Alice decides to be informed about whether she has fulfilled the requirements to get the master degree. Alice uses her PDA to make the appropriate request to the secretary office. The secretary office should authorize Alice for this request due to statements 1, 3, 4, 5 and the information that the office maintains. Thus, the secretary office should proceed in processing the request. The information about Alice (passed all lessons and presented master thesis) leads to a positive answer for her request. Therefore, Alice should receive a positive reply.

Trudy decides to be informed about whether she has fulfilled the requirements to get the master degree. Trudy uses her PDA to make the appropriate request to the secretary office. The secretary office should not authorize Trudy for this request due to statements 1, 3, 4, 5 and the information that the office maintains. Thus, the

secretary office should not proceed in processing the request. Therefore, Trudy should receive neither a negative nor a positive reply.

Professor Antoniou needs a classroom to make a presentation and decides to be informed about whether classroom “RA201” is available at 5 o’clock. The professor uses his PDA to make the appropriate request to the secretary office. The secretary office should authorize Antoniou for this request due to statement 6 and the information that the office maintains. Thus, the secretary office should proceed in processing the request. The information about classroom “RA201” leads to a negative answer for his request. Therefore, the professor should receive a negative reply.

Administrator Smith decides to be informed about whether there is enough memory space left in the computer of the secretary office. The administrator uses his PDA to make the appropriate request to the secretary office. The secretary office should authorize Smith for this request due to statement 7 and the information that the office maintains. Thus, the secretary office should proceed in processing the request. The information about the computer leads to a positive answer for his request. Therefore, the administrator should receive a positive reply.

This scenario makes also clear the demand for a flexible and declarative authorization language for Ambient Intelligence environments that is able to support policies of this kind.

1.4 Approach

Our work builds on previous work [31, 32, 33] for distributed contextual reasoning in ambient intelligence environments, called Contextual Defeasible Logic (CDL). Although CDL provides a flexible language for reasoning about context in distributed environments, it does not address authorization issues. In this work we propose the *Distributed Environment Authorization Language* (*DEAL*), a language that enables the specification of authorization policies in Ambient Intelligence environments. The implementation of the language on top of CDL framework provides two main advantages. Firstly, CDL is enriched with the ability to address authorization issues of intelligent devices in AmI environments. Secondly, CDL enables DEAL to specify authorization policies that require integration of knowledge from heterogeneous information sources. The language has rich expressive power by supporting negative

authorizations, rule priorities, hierarchical category authorizations and nonmonotonic reasoning.

1.5 Thesis Contribution

The theoretical contribution of this work is summarized in the following points.

- We provide a thorough description of the authorization problem in Ambient Intelligence environments by introducing the basic notions of the problem.
- We analyze the desirable features of a language that addresses the authorization problem in Ambient Intelligence environments. More specifically, we describe the characteristics of negative authorization, rule priorities, hierarchical category authorization and nonmonotonic reasoning.
- We present the Distributed Environment Authorization Language (DEAL). DEAL is a formal, high level, logic based language that is able to handle authorization issues of Ambient Intelligence environments. We provide the language's syntax and semantics.
- We extend the basic algorithm of CDL in order to support DEAL policies.

The practical contribution of this work is summarized in the following points.

- We provide the implementation of DEAL rules and the basic extensions to CDL framework in order to fully support DEAL language.
- We implement real application scenarios that aim on illustrating the expressive power of DEAL in association with CDL for addressing the authorization issues of Ambient Intelligence systems.

1.6 Thesis Organization

The rest of this Thesis is organized as follows:

Chapter 2 presents basic concepts involved in the authorization problem in Ambient Intelligence environments. More specifically, it introduces the notions of

request-pair, grantor, grantee, service, authorization conflict, authorization policy, authorization rule and conflicting rules.

Chapter 3 describes the desirable features of a powerful authorization language in Ambient Intelligence environments. This chapter analyzes the importance of negative authorization, rule priorities, hierarchical category authorization and nonmonotonic reasoning as characteristics of a logic-based authorization language for AmI environments.

Chapter 4 presents related research on the authorization problem. This chapter describes the limitations of other logic-based and non-logic based approaches that aim on addressing the authorization problem. The logic-based approaches are distinguished into centralized and decentralized.

Chapter 5 provides background information on contextual reasoning. Firstly, it presents Defeasible Logic and the concept of Multi-Context systems. Secondly, it describes Contextual Defeasible Logic (CDL) as a fully distributed approach for contextual reasoning in Ambient Intelligence environments and provides the representation model.

Chapter 6 introduces the Distributed Environment Authorization Language (DEAL) that aims on providing a powerful logic-based approach for addressing authorization issues in Ambient Intelligence environments. First of all, the chapter presents the syntax of DEAL by providing the alphabet, rules and characteristics of the language through examples. Secondly, it illustrates the semantics of the language through transformation to Defeasible Logic. Thirdly, it describes the appropriate extensions to the basic CDL algorithm in order to support DEAL policies. Finally, the chapter provides the implementation of the motivating scenarios using DEAL policies.

Chapter 7 summarizes the main points of the thesis and discusses potential extensions of this work.

Basic Concepts of the Authorization Problem

Authorization is an important part of the access control process in distributed environments. In this chapter we describe basic concepts that are associated with the authorization problem in ambient intelligence environments.

2.1 Request-Pair

Intelligent devices of AmI environments act as autonomous entities by sending and receiving requests from other entities. The requests aim on consuming services that the entities provide. The pair that consists of the requester entity and the requested service is usually called request-pair. Figure 2.1 presents an example where the request-pair consists of the requester *Bob* and the requested service *translateToGreek*. The request-pair should be included in every received message in order to enforce authorization control on the receiver entity.

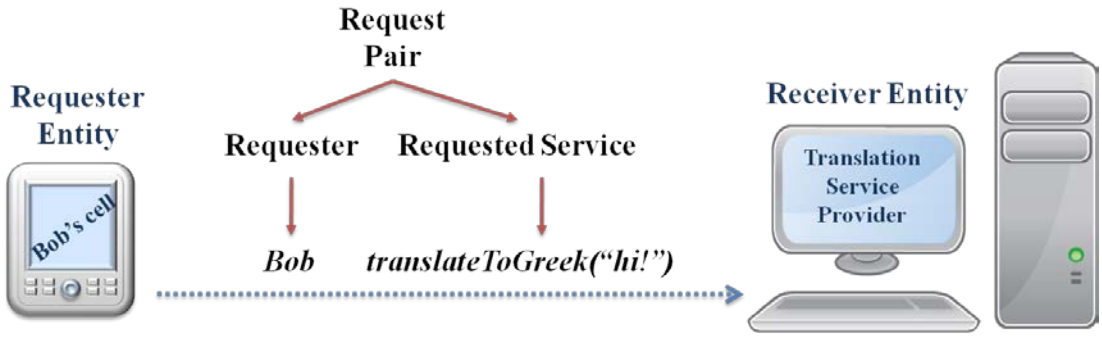

Figure 2.1: The request-pair of a simple Ambient Intelligence example.

2.2 Authorization

Authorization issues arise when an Ambient Intelligence entity receives a request that is either sent from an intelligent device or perceived through human interaction. An authorization plain statement that is usually called authorization expresses either permission (*positive authorization*) or denial (*negative authorization*) for a particular request-pair. The three basic concepts that are usually involved in an authorization

are the *service*, the *grantor* and the *grantee*. These concepts answer the three following questions respectively:

- What does the authorization concern?
- Who is giving the authorization?
- Whom is the authorization given to?

An example of an authorization in the language which is proposed in [29] is depicted in figure 2.2. The authorization defines that the grantor Alice expresses a denial to the grantee Bob for consuming the service that provides access to the file with personal information. The sign +/- denotes either a permission or a denial for providing a service to the grantee

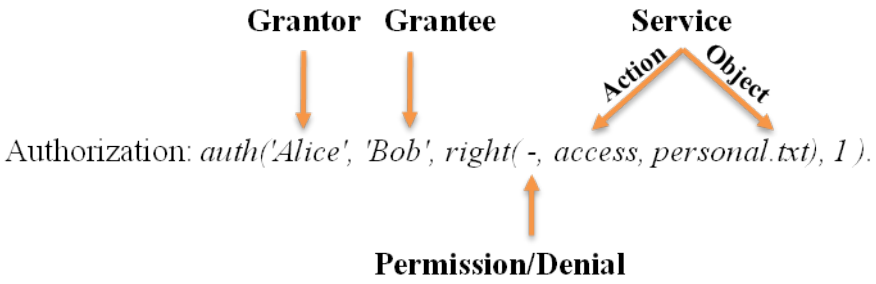

Figure 2.2: An authorization example in language [29].

2.2.1 Service

In an ambient Intelligence environment an AmI entity provides a service in order to be consumed by other AmI entities. The request for consuming a service is usually represented in the form of a query. Two examples of service requests are presented in figure 2.3.

Service request example 1	**Service request example 2**
A query for getting information for the weather state on Friday:	A query for getting permission to read a file with grades:
weather(good, Friday)?	*read(grades.txt)?*

Figure 2.3: Two query examples of service requests.

In [23, 24, 28, 30] the concept of service is decomposed into two additional concepts, an *action* and an *object*, that represent the right to perform an action to a resource object. Example 2 of figure 2.3 is associated with the right to perform the action "read" to the resource "grades.txt". However, the service concept in ambient intelligence environments is a generic notion that depends on the application demands and cannot always be decomposed into an action to an object, such as the example 1 of figure 2.3.

2.2.2 Grantor

Grantor (or *authorizer*) refers to the entity that provides the authorization for a specific request-pair. In an Ambient Intelligence environment it can have the form of a software component of an autonomous intelligent device or of an individual user. Sometimes the grantor is omitted from the specification of authorizations. This usually happens when it is assumed that the grantor is always the local system.

2.2.3 Grantee

Grantee refers to the entity that receives the authorization for consuming a specific service. In an Ambient Intelligence environment it can have the form of a software component of an autonomous intelligent device or of an individual user. Moreover, the grantee of an authorization may refer to a group of entities indicating that every entity of the group receives the same authorization.

2.3 Authorization Conflict

Ambient Intelligence scenarios may include only positive, only negative or both authorizations types. Scenarios that involve positive and negative authorizations may potentially lead to authorization conflicts. An *authorization conflict* describes a problematic state where a positive and a negative authorization exist for the same grantor, grantee and service instances. The positive authorization expresses the permission and the negative authorization expresses the denial of the grantor for providing the service to the grantee. However, the aim of an authorization system as a

grantor concept is to finally conclude either permission (positive authorization) or denial (negative authorization) for a particular service and grantee that refer to a specific request-pair.

The resolution of an authorization conflict requires the specification of a preference among the contradictory authorizations. The basic idea is that the non preferred authorization is nullified, thus the conflict is resolved. Three authorization conflict resolution options could be taken:

- *Denial-preference:* The negative authorization is preferred over the positive, thus the positive is nullified. Therefore, the resolution of the conflict leads to the conclusion that the grantor denies to provide the service to the grantee.
- *Permission-preference:* The positive authorization is preferred over the negative, thus the negative is nullified. Therefore, the resolution of the conflict leads to the conclusion that the grantor permits the grantee to consume the service.
- *No-Preference:* None authorization is preferred, thus both authorizations are nullified. Therefore, no conclusion can be derived about whether the grantor permits or denies the grantee to consume the service. The final outcome in this case may be specified according to the needs of the particular system. For instance, the system may handle this case as an error or it may ask for additional information in order to reach as decision.

An authorization conflict can be viewed as a specific type of a *knowledge conflict*. A knowledge conflict refers to pairs of data elements that meet specific requirements which express contradictory knowledge. An instance of a knowledge conflict is a particular pair of data elements that fulfill the specified requirements. An authorization conflict is a specific type of knowledge conflict where the contradictory data elements correspond to a permission and a denial of a grantor for providing a service to a grantee.

The authorization conflict resolution options that apply to an authorization conflict can be generalized for the knowledge conflict. More specifically, given a pair of contradictory data elements that form an instance of a knowledge conflict there are

three knowledge conflict resolution options which can be applied, the *first-preference*, the *second-preference* and the *no-preference*. The first-preference option indicates that the first data element of the pair is preferred over the second. On the other hand, the second-preference option indicates that the second data element of the pair is preferred over the first. Lastly, the no-preference option indicates that the knowledge conflict is resolved without preferring any of the contradictory data elements. The specific semantics of these conflict resolution options depend on the particular type of knowledge conflict on which they are applied.

In a particular environment there can be many types of knowledge conflicts according to what is considered contradictory knowledge. The specification of a type of knowledge conflict is an application dependant subject. In other words, the pairs of data elements that are considered contradictory depend on the particular application domain. In authorization applications, contradictory data elements are usually elements that form an authorization conflict. However, authorization applications may require also the specification of other types of knowledge conflicts.

2.4 Authorization Policy

Authorization policy is defined as a set of authorizations and conditions under which they are concluded. An authorization policy describes when a requester should be provided or denied a specific service.

The problem of specifying an authorization policy for a system's resources can be viewed as a knowledge representation (KR) problem. Logic-based approaches have been proven very successful in knowledge representation because they offer significant advantages such as simplicity, flexibility, formality, expressivity and modularity as described in [61]. Therefore, logic-based approaches are commonly used for expressing authorization policies. In logic-based approaches, an authorization policy is defined as a set of logical rules. A logical rule that participates in the process of authorization is called *authorization rule*. An authorization rule can be either a *final* rule which concludes to an authorization or an *intermediate rule* which specifies an intermediate conclusion which is not an authorization. Ambient Intelligence authorization scenarios usually require multiple authorization rules in order to define permissions and denials under different circumstances. This is due to the many

different aspects of context and the many different possible states of the environment or the system, which possibly have to be taken into account.

In KR logic-based approaches, a pair of rules is considered *conflicting* (or *contradictory*) if their conditions can be simultaneously satisfied and their conclusion instances can form a knowledge conflict instance. In other words, two rules are considered conflicting if they potentially lead to a knowledge conflict. A knowledge conflict that is caused by two conflicting rules is also called *rule conflict*.

An example of a pair of conflicting rules is depicted in figure 2.4. The example includes two final authorization rules. The rules may potentially lead to an authorization conflict. The rules are specified in the language of [29]. The study in [29] describes a KR logic-based approach which focuses on the authorization problem.

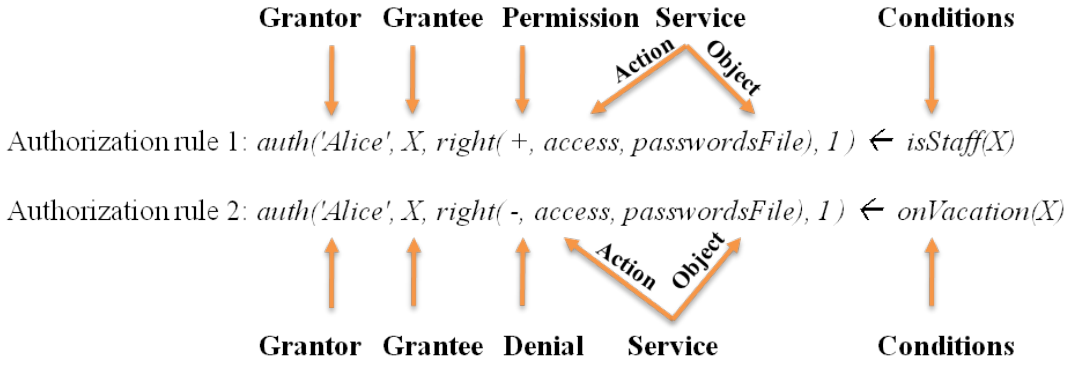

Figure 2.4: An example of conflicting rules in language of [29].

Moreover, given a contradictory pair of rules, there is a conflict resolution approach which is based on rule preference, that aims to resolve every potential rule conflict caused by these rules in the same way. This approach includes three conflict resolution options, the *firstRule-preference*, the *secondRule-preference* and the *noRule-preference*. The firstRule-preference option resolves every potential rule conflict by deriving always the conclusion of the first rule while the conclusion of the second is blocked (not derived). On the other hand, the secondRule-preference approach resolves every potential rule conflict by deriving always the conclusion of the second rule while the conclusion of the first is blocked. Lastly, the noRule-preference approach resolves every potential rule conflict by blocking both rule conclusions. In this approach, neither of the rule conclusions can be derived. Thus,

every rule conflict is avoided.

Furthermore, given a contradictory pair of rules and a rule preference for conflict resolution. The rule that is preferred is called *superior* while the other rule is called *inferior*. In case where no rule preference is specified (noRule-preference approach) to the contradictory rule pair, the rules are called *neutral* regarding rule preference.

Desirable Characteristics of an Authorization Language

Ambient intelligence systems require an authorization language with specific characteristics in order to support expressive authorization policies. In this chapter we present the desirable features of a powerful authorization language for Ambient Intelligence environments and argue their usefulness in detail.

The desirable characteristics of an Ambient Intelligence authorization language are listed below:

- Negative authorization
- Rule priorities
- Hierarchical category authorization
- Nonmonotonic reasoning

3.1 Negative Authorization

A negative authorization expresses the denial of a grantor to provide a service to another to a grantee. The specification of negative authorizations is required in many common Ambient Intelligence scenarios that involve blocking of specific request-pairs. For example, many scenarios include services that expose private information or permit access to private resources. Requests that aim to consume this type of services are considered private. In these scenarios it must somehow be declared negative authorization to specific requesters for private requests.

The AmI entities and the authorization policy of a simple Ambient Intelligence scenario that involves negative authorization are provided below.

Ambient Intelligence entities:

(a) Individual user: Professor Antoniou owns an intelligent mobile phone;

(b) Intelligent device: His phone maintains an automatic system for answering requests about passing a recent exam;

Special cases of students require further consideration by the professor before answering. Thus, the professor wants to specify the following policy for any incoming requests about passing the recent exam.

Authorization policy:

1. *A denial to answer requests from his two students Bob and Alice because they probably cheated.*
2. *A denial to answer requests if the requester is a student and has exceeded time limit during the exam because the grade must be decreased.*

The authorization policy of the scenario is depicted in a matrix in figure 3.1.

AUTHORIZATION POLICY				
Authorization				**Conditions**
Authorization Type	Grantor	Grantee	Service	
Denial	*Antoniou*	*Bob*	*Pass Exam*	-
Denial	*Antoniou*	*Alice*	*Pass Exam*	-
Denial	*Antoniou*	*Any*	*Pass Exam*	*Grantee is student and exceeded time limit.*

Figure 3.1: An authorization policy with negative authorizations.

This scenario includes professor Antoniou as an individual user and a mobile phone as an intelligent device. The authorization policy requires two negative authorizations by the grantor professor Antoniou to the grantees Bob and Alice for the service of answering exam grades requests. Moreover, the policy requires a rule that concludes a negative authorization for students that have exceeded time limit. An authorization language should be able to represent negative authorizations in order to easily support simple scenarios like this. Therefore, negative authorization is a desirable feature of an authorization language for Ambient Intelligence environments.

3.2 Rule Priorities

Rule priorities is a feature that enables the specification of a priority relation to a contradictory pair of rules. The priority relation can be used to denote a preference to the contradictory rules. Thus, rule preference for conflict resolution (described in

section 2.4) can be expressed with rule priorities. In other words, the priority relation can be used to specify either the firstRule-preference or the secondRule-preference conflict resolution option that is able to resolve every potential rule conflict caused by the contradictory rules. This feature is useful in many common AmI scenarios that involve multiple authorization rules which may potentially lead to inconsistencies (authorization conflicts). The authorization conflicts in these scenarios can be easily avoided by specifying a priority on each contradictory rule pair.

An example of a simple Ambient Intelligence scenario that requires rule priorities is presented below. This scenario takes place in a company which includes a private area. Mr. Smith as the manager of the company controls the access to this area. The context information flow of the scenario is depicted in figure 3.2.

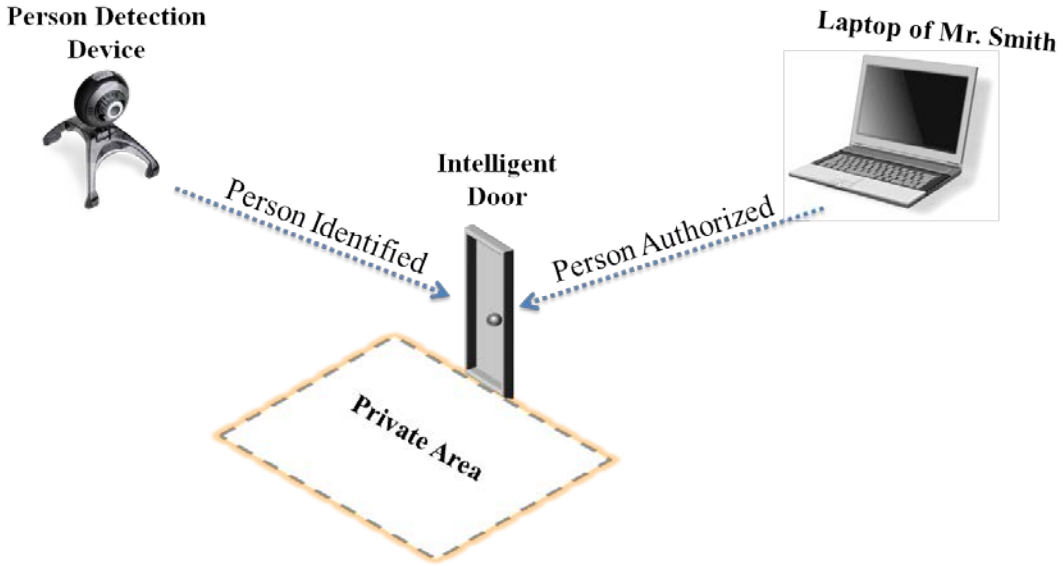

Figure 3.2: Context information flow in the scenario

The AmI entities and the authorization policy of the scenario are provided below.

Ambient Intelligence entities:

(a) Individual user: The manager of the company Mr. Smith;

(b) Intelligent device: A laptop used by Mr. Smith maintains an intelligent system for specifying access policies to private area (PA);

(c) *Intelligent device: A person detection device indentifies any individual that requests access to PA;*

(d) *Intelligent device: An intelligent door lock mechanism consults the person detection device and Smith's laptop preferences in order to control the entrance to PA;*

Authorization policy: Mr. Smith wants to specify the following four rules in the below preference order in case of conflicts.

1. *A permission for accessing PA is granted to a person if the person is the manager;*
2. *A denial for accessing PA is specified to a person if the person is a fired employee;*
3. *A permission for accessing PA is granted to a person if the person is a submanager;*
4. *A permission for accessing PA is granted to a person if the person is a system administrator;*

The authorization policy of the scenario is depicted in a matrix in figure 3.3.

AUTHORIZATION POLICY				
Authorization				**Conditions**
Authorization Type	Grantor	Grantee	Service	
1. Permission	*Smith*	*Any*	*Access PA*	*Grantee is the manager*
2. Denial	*Smith*	*Any*	*Access PA*	*Grantee is a fired employee*
3. Permission	*Smith*	*Any*	*Access PA*	*Grantee is a submanager*
4. Permission	*Smith*	*Any*	*Access PA*	*Grantee is a system administrator*

Figure 3.3: An authorization policy that requires rule priorities.

The authorization policy involves the conflicting rule pairs {(1, 2), (2, 3), (2, 4)} and requires the specification of three priorities relation on these pairs in order to implement the manager's preference in the rules. More specifically, the first priority relation should express that rule 1 is preferred than rule 2. The second priority relation should define that rule 2 is preferred than rule 3 and the third priority relation should

specify that rule 2 is preferred than rule 4. This kind of scenarios that involve many and possibly conflicting authorization rules require an authorization language that supports priorities relations on policy rules. Thus, rule priorities is a desirable feature for an authorization language in Ambient Intelligence environments in order to easily specify consistent policies of multiple authorization rules.

3.3 Hierarchical Category Authorization

Ambient Intelligent entities can be grouped into hierarchical categories. A category expresses a common property for the belonging entities. An AmI entity may belong to many different categories. A category may also belong to many other categories as their specialization organizing thus hierarchies. An example of hierarchical categories of users is presented in figure 3.4.

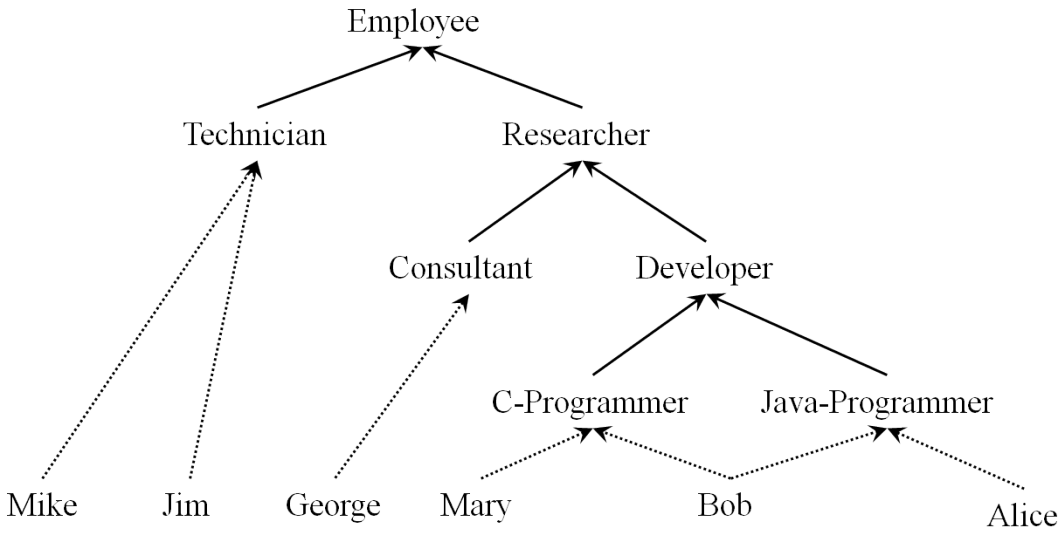

Figure 3.4: An example of user hierarchical categories.

Ambient Intelligent services can also be grouped into hierarchical categories. An example of hierarchical categories of services is presented in figure 3.5. The root element of the structure is the category "dataService".

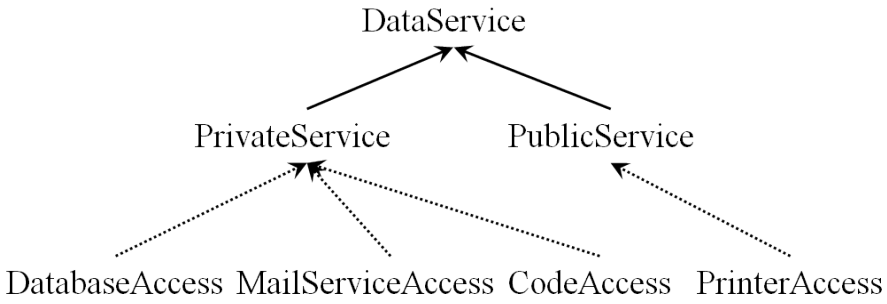

Figure 3.5: An example of service hierarchical categories.

Moreover, when services are decomposed into action and object concepts, these additional concepts can also be structured into hierarchical categories. Two examples that represent hierarchical categories of actions and hierarchical categories of objects are shown in figure 3.6 and figure 3.7 respectively.

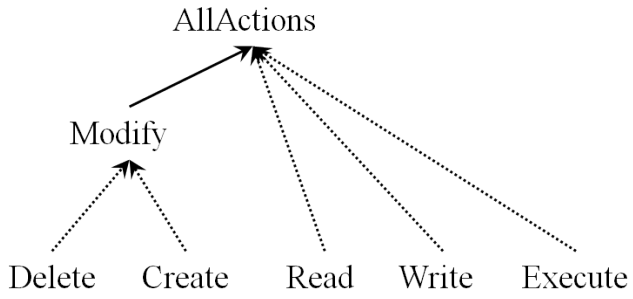

Figure 3.6: An example of action hierarchical categories.

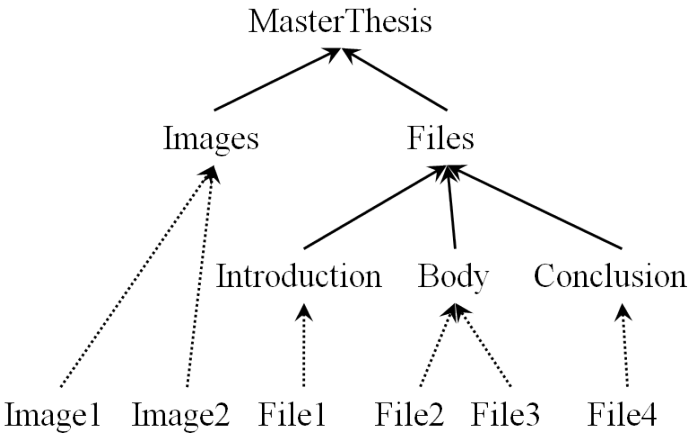

Figure 3.7: An example of object hierarchical categories.

The ability to represent hierarchical categories is a feature that enables Ambient Intelligence systems to define structured services, actions, objects and AmI entities. Furthermore, the ability to specify inherited authorizations among hierarchical categories simplifies the process of authorization. The main simplification is that an authorization can be related with a category instead of declaring the same authorization for every element of the category. For example, a permission for accessing a project can be associated with the category researcher of figure 3.4 instead of authorizing Bob, Alice, Mary and George separately. An authorization language should be able to represent hierarchical categories and inherited authorizations among them in order to simplify the authorization task. Therefore hierarchical category authorization is a desirable feature of an authorization language in Ambient Intelligence environments.

3.4 Nonmonotonic Reasoning

Nonmonotonic reasoning is reasoning based on the absence of information and was developed to model commonsense reasoning used by humans. Nonmonotonic reasoning is especially appropriate for specifying authorization policies in Ambient Intelligence environments. The information in Ambient Intelligence systems may be incomplete or changing due to the open and dynamic nature of AmI environments. The development of nonmonotonic reasoning provides formal methods that enable Ambient Intelligence systems to handle incomplete or changing information and derive authorization conclusions that in the presence of future information may be withdrawn.

Many Ambient Intelligence authorization scenarios involve nonmonotonic policies. An example of a simple AmI authorization scenario that requires nonmonotonic reasoning is presented below. This scenario includes an individual user (painter Nick) and an intelligent device (laptop). The AmI entities and the authorization policy of the scenario are provided below.

Ambient Intelligence entities:

(a) Individual user: Nick is a painter that takes photos of his finished art and uploads them on a photo gallery in his laptop;

(b) Intelligent device: A laptop used by Nick maintains an intelligent system for answering requests about accessing photos;

Authorization policy: Nick wants to enforce the following nonmonotonic authorization policy.

1. *A permission for accessing a photo of his gallery is granted to a person if there is no information that the depicted painting is sold;*
2. *A denial for accessing a photo of his gallery is specified to a person if the depicted painting is sold;*

The authorization policy of the scenario is depicted in a matrix in figure 3.8.

AUTHORIZATION POLICY				
Authorization				**Conditions**
Authorization Type	Grantor	Grantee	Service	
1. Permission	*Nick*	*Any*	*Access Photo*	*Absence of selling information about the photo*
2. Denial	*Nick*	*Any*	*Access Photo*	*Presence of selling information about the photo*

Figure 3.8: An authorization policy with nonmonotonic reasoning.

The first rule of the policy includes nonmonotonic features. More specifically, it expresses that permission is concluded for any request, about accessing a photo, only in the absence of information about sale. The second rule specifies a denial for accessing a photo in the presence of information about sale. Note that the two rules cannot lead to an authorization conflict because their conditions cannot be simultaneously satisfied. Thus, this policy does not require conflict resolution with rule priorities. The nonmotonicity of the policy derives from the fact that future information about sale may withdraw previous permissions. An authorization language should support nonmonotonic reasoning in order to easily specify policies like this. Therefore, nonmonotonic reasoning is a desirable feature of an authorization language in Ambient Intelligence environments.

Related Work

Over the past twenty years, there have been proposed several authorization approaches, both logic and non logic based, for distributed environments. In this chapter we present the approaches that are related to our work and we provide their limitations thoroughly.

In order to describe the limitations of the authorization approaches we distinguish the authorization policies into distributed and centralized. A centralized authorization policy involves only the local information of the authorizer while a distributed authorization policy requires information from external third-party entities. A distributed authorization is an authorization that is concluded from a distributed authorization policy.

An example of a distributed authorization would be the second authorization rule of the policy in the motivating scenario of section 1.3.1, if the information that refers to the doctor permission, required a communication with the doctor, instead of being maintained locally in the management office. For simplicity reasons, we consider it local information.

Distributed authorizations are required in many cases. For instance, in some cases the processing of the authorization for a particular request-pair may be divided into several sub-processes. It is possible that an entity alone cannot deal with all the sub-processes thus different entities of the environment are specialized for different sub-processes. These cases require distributed authorization policies that involve information from external specialized entities.

Moreover distributed authorizations are required in cases where the authorizer wants for some reason (e.g. lack of information or decision confirmation) to consult third-party entities about authorizing a request-pair. We call the process of taking into account a third-party opinion, about authorizing a particular request-pair, *authorization consultation*. In these cases there are arising issues of conflict resolution (how to handle conflicting third-party opinions). An approach for conflict resolution could be enforced based on a total or partial ordering of the third-parties regarding their reliability. However the enforcement of a specific conflict resolution approach is strictly dependant on the application requirements.

Finally distributed authorizations are required in cases where the authorizer has for some reason (e.g. to decrease his work load) delegated the right of authorizing specific grantee-service pairs to other third-party entities that he trusts. The process of transferring the right of authorization to a third-party entity is called *authorization delegation*. In these cases where the authorizer has made authorization delegations he should specify distributed authorization policies that involve information from the external delegated entities in order to make authorization decisions (for request-pairs that are related with the authorization delegations). In addition these cases that involve authorization delegations may lead to authorization conflicts because it is implied that the authorizer will agree with the "beliefs" of the delegated third-parties (possible presence of conflicting third-party opinions).

The difference between *authorization consultation* and *authorization delegation* is that the second indicates that the involved third-party entity has been given the right to make authorization decisions while the first one doesn't. Furthermore, authorization delegations are related with large systems that require decentralization. In centralized systems only one entity has the privilege to make important decisions (e.g. authorizations). This entity orders the other entities to make non-essential tasks. On the other hand, in decentralized systems the right to make important decisions can be transferred from the most "privileged" entities (that have more jurisdictions) to the "unprivileged" entities.

In open and dynamic distributed environments, we distinguish two different approaches for the exchange of knowledge with external entities. An authorization framework can adopt either of them in order to support distributed authorizations. These approaches are listed below.

- *Connection-based* approach.
- *Credential-based* approach.

The first approach is based on runtime communications and information gathering from the third-party entities. The authorizer must establish connections with the external entities that he wishes to communicate in order to receive information. A framework that supports distributed authorizations by adopting this approach should be able to represent the information that is gathered from the external entities in the specification of the authorization policies. Moreover, the framework should provide

mechanisms that would implement the required connections and information exchange that is specified in the authorization policies.

The second approach is based on credentials. Credentials represent knowledge in specific file forms that are issued from entities in the environment. Credentials may contain simple facts such as “Bob is a student in the University of Crete” or more complicated policy statements. The key point in this method is that the authorizer must somehow receive credentials which include information from third parties in order to expand his knowledge. The credentials are usually provided by the requester, either together with his request or later according to the communication protocol, in order for his request to be authorized (by the authorizer). A framework that supports distributed authorizations by adopting this approach should be able to represent the information that is gathered from the third-party entities in the specification of the authorization policies. Moreover, the framework should provide mechanisms that would extract the knowledge from valid credentials into the local knowledge of the authorizer.

Both approaches have advantages and disadvantages. The connection-based approach is a direct approach since the authorizer must establish a direct connection with the third-party entity that maintains the required information. Moreover, this approach is more dynamic and flexible since it provides runtime third-party information flow that can be specified in the authorization policies. The disadvantage of this approach is that it is more time-demanding because the exchange of knowledge with external entities requires additional time for the third-party communications. On the other hand, the credential-based approach requires only the process of extracting the credential information into the local authorizer knowledge (in order to achieve the information exchange with third-party entities). This approach is indirect since the authorizer receives the required information (in the form of credentials) usually from the requester entity which is not related with the third-party entity that issued them. The disadvantage of this approach is that it is more static in the sense that it does not provide "fresh" information that is gathered during the request processing since credentials may be issued any time earlier. Moreover, the indirect static nature of the second approach includes more risks on the secure information flow.

The most suitable approach for supporting distributed authorizations for a specific system depends on the specific requirements of the application. It may even be a hybrid combination of the two approaches.

4.1 Non logic-based Authorization Approaches

The trust-management approach, which was initially proposed by Blaze et al. in [11] and is focused on the credential-based method for distributed authorization, has received a great attention by many researchers [12, 14, 26, 28, 41]. In the trust management approach a requester submits a request, possibly supported by a set of credentials, to an authorizer who controls the requested service. The authorizer then decides whether to authorize this request pair by answering the authorization question:

> *"Should this request pair be authorized based on the submitted credentials, my knowledge and authorization policy?"*

On the other hand, we could address the submitted request, which is supported with credentials, as two different requests. The first one would be a request for the credentials to be accepted by the authorizer that would lead to upgrading his local knowledge and policy while the second one would be the initial request. This approach would simplify the authorization question in the following:

> *"Should this request pair be authorized based on my knowledge and authorization policy?"*

The first request would trigger the authorization question on the authorizer side in order to be answered. The authorization rules for this request may be associated with conditions for validity of the credentials. The second request would trigger the authorization question on the upgraded authorizer context. In our approach we concentrate on the specification of expressive authorization policies and don't deal with credentials, because we view them as additional requests for services which expand the authorizer's knowledge and policy.

The first attempts towards a trust management system where the following frameworks, the PolicyMaker [11, 12] developed in 1996-1998, the REFEREE [15] developed in 1997, the Keynote [13, 14] developed in 1999 and the SPKI/ SDSI [16-20] developed in 1996-1999.

PolicyMaker was the first trust management system developed by Blaze, Feigenbaum, and Lacy, in the original paper in which the notion of Trust

Management was introduced. PolicyMaker's compliance-checking algorithm was later fleshed out in [12]. In PolicyMaker, policies and credentials together are called "assertions". An assertion is a pair (*f, s*), where *s* is the source of authority (i.e., the issuer of this assertion), and *f* is a program describing the nature of the authority being granted as well as the party or parties to whom the authority is being granted. Assertions can be written in any programming language that can be "safely" interpreted by a local environment. PolicyMaker is quite expressive in the sense that one can code up complex policies. More details about PolicyMaker are avaliable in [11, 12, 14].

REFEREE is similar to PolicyMaker because it also allows arbitrary programming to be used in credentials and policies. However, none of these approaches provides declarative semantics since they allow credentials and policies to contain programs in procedural languages. Moreover they don't support negative authorization which is considered a basic desirable feature for an authorization language in Ambient Intelligence environments.

Keynote is the second generation of trust management systems and was designed according to the same principles as PolicyMaker. Instead of writing policy and credentials in a general-purpose procedural language, it adopts a specific expression language. KeyNote provides declarative semantics by giving a procedure to answer whether a specific request pair should be authorized given a set of credentials. However, neither this approach supports negative authorization.

SPKI (Simple Public Key Infrastructure) and SDSI (Simple Distributed Security Infrastructure) were developed independently. SDSI was originally designed by Rivest and Lampson [20]. SPKI was originally designed by Ellison [18]. Both of them were motivated by the in-adequacy of public-key infrastructures based on global name hierarchies, such as X.509 [43] and Privacy Enhanced Mail (PEM) [42]. Later, SPKI and SDSI merged into a collaborative effort, SPKI/SDSI 2.0 in [16, 17, 19, 44]. SPKI/SDSI 2.0 has two kinds of certificates, name-definition certificates and authorization certificates. A name certificate binds a local name to a principal or a more complex name. However, it is pointed out in [26] that the collaborative effort lacks basic expressive authorization features such as conjunction of attributes and attributes with fields. An authorization policy that is based on conjunction of attributes is "A hospital gives special permissions to anyone who is both a physician and a manager" and another that is based on attributes with fields is "A hospital

allows an entity to access the records of a patient if the entity is the physician of the patient" as presented in [26].

More sophisticated approaches towards an authorization system that are based in logic programming are described in the following section.

4.2 Logic-based Authorization Approaches

In this section we present approaches of [21-30, 45, 46] research that use logic programming to specify authorizations. Logic-based authorization methodology is a very flexible and declarative approach that achieves separation of authorization policies from implementation mechanisms which has long been recognized as a fundamental tenet in the design of an authorization system. Moreover, this approach provides policies with precise semantics.

The first research that aimed on the specification of authorizations based on logic languages was the work of Woo and Lam [45] in 1993. Their work points out the need for flexibility and extensibility in the specification of authorization policies. The Woo and Lam research illustrates the benefits of abstracting from low level authorization triples to a high level logic-based authorization language. The authorization model of their approach is depicted in figure 4.1.

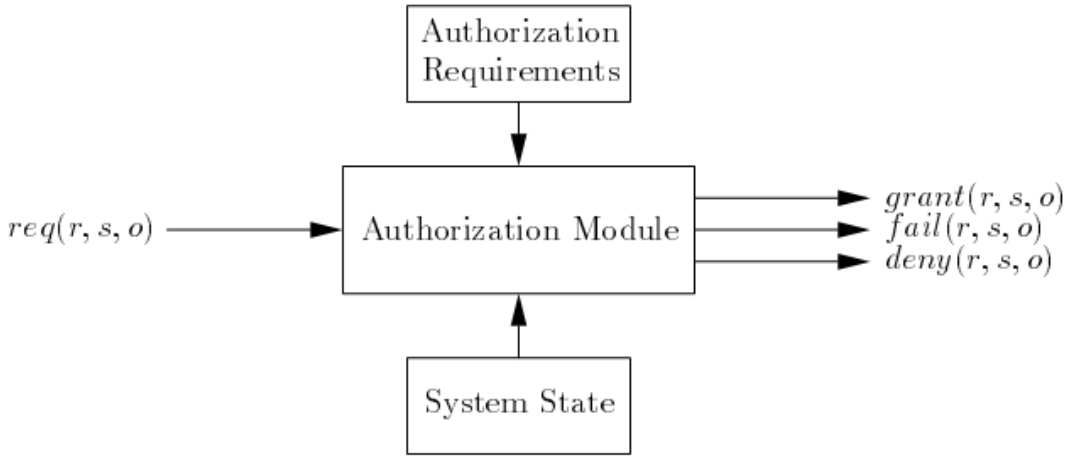

Figure 4.1: Model of authorization in Woo, Lam approach [45].

In the authorization model a subject refers to a requester entity and an object to a specific resource. The authorization model is based on the following process. Before a subject *s* can perform a particular access *r* on an object *o*, *s* must first obtain the access right *r* for *o*. Subject *s* does so by submitting a request of the form *req(r; s; o)* to the authorization module, which responds with *grant(r; s; o)*, *deny(r; s; o)* or *fail(r; s; o)*. A *grant(r; s; o)* is returned if the authorization module can determine that *s* is authorized to have *r* access to *o*, while a *deny(r; s; o)* is returned if the authorization module can determine that *s* is denied *r* access to *o*. *A fail(r; s; o)* is returned if the authorization module fails to establish either one of the previous two cases. To make an authorization decision, the authorization module consults the authorization requirements and the system state. The system state is needed for authorization requirements that contain system state variables as parameters. A more detailed description is provided in [45].

4.2.1 Centralized Authorization Approaches

In 1997-2003 there have been proposed several logic-based approaches that aim on the specification of authorization policies [21, 22, 23, 24, 46].

In [21, 22, 24], Jajodia et al. proposed the Flexible Authorization Framework (FAF) that it can be used to specify different access control policies that can all coexist in the same system and be enforced by the same security server. FAF incorporates an authorization specification logic language (ASL) which can be used to encode the system security needs. ASL supports negative authorizations, hierarchical category authorization and seems to be able to express nonmonotonic reasoning by the use of the symbol ¬ (used as negation as failure). However, the language does not support rule priorities for conflict resolution that we consider a basic feature of an authorization logic-based language.

In [23, 46] Bertino et al. proposed a logic formalism for expressing authorization policies. Although the formalism is rich enough to express hierarchical category authorization, negative authorizations (through the use of true negation) and nonmonotonic reasoning (through the use of negation as failure) it does not support rule priorities.

In conclusion, the approaches in [21-24, 46] are able to specify multiple authorization policies and provide formal semantics, yet they don’t support rule

priorities which we consider a desirable feature for conflict resolution. Moreover these works do not address distributed authorizations and are rather focused on centralized systems.

4.2.2 Decentralized Authorization Approaches

In this section we will describe several logic-based approaches that are based on decentralized environments [25-30] and provide rich expressive power.

The approaches in [25, 26] proposed by Li et al. are able to support distributed authorizations and adopt the credential-based approach. More specifically [25] proposes RT framework, a family of role-based trust management languages for representing policies and credentials. The semantic foundation of RT is DATALOG with constraints, which enables RT to express authorization of structured resources and separation of duty policies. In addition the approach in [26] presents Delegation Logic (which was previously presented in [68, 69]) that is able to specify complex principles including k-out-of -n structures and delegation depth using also DATALOG as the semantic foundation. Yet, neither of these approaches supports negative authorization, nonmonotonic reasoning and rule priorities.

The approach in [27] presents the nonmonotonic version of Delegation Logic and the approach in [30] proposed by Liu et al. presents the nonmonotonic framework FACL4DE. Both approaches are able to express negative authorization, nonmonotonic reasoning (through negation as failure using symbol ~) and rule priorities. However, neither of these approaches supports hierarchical category authorization.

Finally, in [28,29] Wang et al. proposed the language AL. Although AL has rich expressive power by supporting negative authorization, nonmonotonic reasoning (through negation as failure) and hierarchical category authorization, it is not able to specify rule priorities.

In conclusion, logic-based approaches have been proven very successful in specifying authorizations. However, most existing authorization logic-based approaches don’t provide the desirable characteristics for an authorization language in Ambient Intelligence environments and thus don't meet the demanding needs of these environments.

Background Information

This chapter provides background information on Defeasible Logic, introduces the notion of Multi-Context Systems and describes Contextual Defeasible Logic and its representation model.

5.1 Defeasible Logic

Defeasible logic is a simple and efficient rule based non-monotonic formalism that was originally created by Donald Nute [47]. A thorough research that is based on the formalism is provided also in [48, 34]. The logic has been extended over the years and several variants have been proposed. The main focus of the logic is to be able to derive conclusions from incomplete and sometimes conflicting information. Thus, the logic was developed to support "tentative" conclusions (defeasible conclusions) and conflict resolution. In case of conflicting information, the logic provides a conflict resolution approach based on priority relations between the contradictory data. In case of incomplete information, the logic is able to express defeasible conclusions which are conclusions that can be withdrawn in the presence of new information.

The representation of the knowledge in Defeasible Theory is based on three concepts, facts, rules and superiority relation. Rules are divided into three categories, strict rules, defeasible rules and defeaters. A description of these concepts is provided below.

- *Facts:* Facts are indisputable statements.
- *Strict Rules:* Strict rules are "classical" rules in the sense that whenever the premises are indisputable (e.g., facts) then so is the conclusion.
- *Defeasible Rules:* Defeasible rules are rules that their conclusions can be defeated by contrary evidence.
- *Defeaters:* A defeater is a rule which its conclusion can be derived if its conditions and the conditions of a contradictory rule are satisfied. In other words, a defeater conclusion cannot be derived without a derived contradictory rule conclusion, even if the conditions of the defeater are satisfied. Their only use is to conditionally prevent other rule conclusions by providing contrary evidence.

- *Superiority Relation:* The superiority relation is a binary relation defined over a pair of conflicting rules. The superiority relation determines which rule conclusion is stronger in case of an arising conflict from these rules.

Defeasible Logic reasoning is “skeptical”. This characteristic derives from the fact that where there is some support for concluding A but also support for concluding the negation of A (¬A), neither of them is concluded and the logic consults the priority relations to resolve the issue. If the support for A has priority over the support for ¬A then A is concluded.

In Defeasible Logic, the derived conclusions (of a derivation process over a set of rules) can be distinguished into two categories, definitely provable and defeasibly provable.

Definitely provable are the conclusions that can be derived using only facts and strict rules. More specifically a conclusion is definitely provable in the three following cases:

- *If it is a fact.*
- *If it is conclusion of a strict rule which its conditions are satisfied by given facts.*
- *If it is conclusion of a strict rule that can be derived by forward chaining process that is based only on strict rules and facts.*

Defeasibly provable conclusions are conclusions that their derivation process may involve defeasible rules. More specifically a conclusion is defeasibly provable in the three following cases:

- *If it is definitely provable.*
- *If it is conclusion of strict or defeasible rule which its conditions are satisfied by given facts (applicable) and all contradictory rules are not applicable.*
- *If it is conclusion of an applicable strict or defeasible rule and all contradictory rules are weaker based on the priority relations among them.*

5.1.1 Proof Theory

Informally, a conclusion q is defeasibly derivable given a defeasible theory D = (F, R, >) when (a) q is a fact; or (b) there is an applicable strict or defeasible rule for q, and either all the rules for q-complementary literals are discarded or every rule for a q-complementary literal is weaker than an applicable rule for q.

Formally, a conclusion of a defeasible theory D is a tagged literal and can have one of the following four forms:

- $+\Delta q$ which is intended to mean that q is a definite consequence of D
- $-\Delta q$ which is intended to mean that we have proved that q is not a de finite consequence of D
- $+\theta q$ which is intended to mean that q is defeasible provable in D
- $-\theta q$ which is intended to mean that we have proved that q is not defeasible provable in D

Provability is based on the concept of a derivation in D [34]. A derivation is a definite sequence P = (P(1), ,P(n)) of tagged literals satisfying the following conditions (P (1..i) denotes the initial part of the sequence P of length i, R^s [q] the set of strict rules that support q and R^d [q] the set of defeasible rules that support q):

$+\Delta$: *If* $P(i+1) = +\Delta q$ *then either*
$q \in F$ *or*
$\exists r \in R^s[q]\ \forall \alpha \in body(r)$: $+\Delta\alpha \in P(1...i)$

$-\Delta$: *If* $P(i+1) = +\Delta q$ *then either*
$\in F$ *and*
$\forall r \in R^s[q]\ \exists \alpha \in body(r)$: $-\Delta\alpha \in P(1...i)$

$+\theta$: *If* $P(i+1) = +\theta q$ *then either*
(1) $+\Delta q \in P(1...i)$ *or*
(2) (2.1) $\exists r \in R^{sd}[q]\ \forall \alpha \in body(r)$:
$+\theta\alpha \in P(1...i)$ *and*

(2.2) $-\Delta \sim q \in P$ (1...i) *and*

(2.3) $\forall s \in R[\sim q]$

(2.3.1) $\exists \alpha \in body(s)$: $-\theta\alpha \in P$ (1...i) *or*

(2.3.2) $\exists t \in R^{sd}[q]$:

$\forall \alpha \in body(t)$: $+\theta\alpha \in P$ (1...i) *and* $t > s$

$-\theta$: *If* $P(i+1) = -\theta q$ *then*

(1) $-\Delta q \in P$ (1...i) *and*

(2) (2.1) $\forall r \in R^{sd}[q]\ \exists \alpha \in body(r)$:

$-\theta\alpha \in P$ (1...i) *or*

(2.2) $+\Delta \sim q \in P$ (1...i) *or*

(2.3) $\exists s \in R[\sim q]$ *such that*

(2.3.1) $\forall \alpha \in body(s)$: $+\theta\alpha \in P$ (1...i) *and*

(2.3.2) $\forall t \in R^{sd}[q]$ *either*

$\exists \alpha \in body(t)$: $-\theta\alpha \in P$ (1:::i) *or* $t \not> s$

Governatori et. al describe in [62] Defeasible Logic and its variants in argumentation theoretic terms. A model theoretic semantics is discussed in [63].

5.2 Multi-Context Systems

A Multi-Context System (based on Bikakis et. al. research [37]) consists of a set of contexts and a set of inference rules (known as mapping or bridge rules) that enable information flow between different contexts. A context can be thought of as a logical theory - a set of axioms and inference rules - that models local knowledge of an agent. Different contexts are expected to use different languages and inference systems, and although each context may be locally consistent, global consistency cannot be required or guaranteed. Reasoning with multiple contexts requires performing two types of reasoning; (a) local reasoning, based on the individual context theories; and (b) distributed reasoning, which combines the knowledge encoded in different local theories using the mappings. The most critical challenges of contextual reasoning are the heterogeneity of local context theories, and the potential conflicts that may arise from the interaction of different contexts through the mappings.

The notions of context and contextual reasoning were first introduced in AI by McCarthy in (1987) [49], as an approach for the problem of generality. In the same paper, he argued that the combination of non-monotonic reasoning and contextual reasoning would constitute an adequate solution to this problem. Since then, two main formalizations have been proposed to formalize context: the Propositional Logic of Context, PLC in [50, 51], and the Multi-Context Systems introduced by Giunchiglia and Serafini in [35], which later became associated with the Local Model Semantics proposed by Ghidini and Giunchiglia in [52]. Multi-Context Systems have been argued to be most adequate with respect to the three properties of contextual reasoning (partiality, approximation, proximity) and shown to be technically more general than PLC in the research of Serafini and Bouquet [53]. This formalism was also the basis of two recent studies that were the first to deploy non-monotonic features in contextual reasoning:

1. the non-monotonic rule-based MCS framework [54] (Roelofsen and Serafini, 2005) which supports default negation in the mapping rules allowing to reason based on the absence of context information.
2. the multi-context variant of Default Logic, ConDL [55] (Brewka et al., 2007) which models bridge relations between different contexts as default rules.

Additionally to the three fundamental dimensions of contextual reasoning (partiality, approximation and perspective) that the generic MCS model inherently supports, both approaches support reasoning with incomplete local information using default negation in the body of the mapping rules. Furthermore, Contextual Default Logic handles the problem of mutually inconsistent information provided by two or more different sources using default mapping rules, and has the additional advantage that is closer to implementation due to the well-studied relation between Default Logic and Logic Programming. However, ConDL does not provide ways to model the quality of imported knowledge, nor preference between different information sources, leaving the conflicts that arise in such cases unresolved.

The use of Multi-Context Systems as a means of specifying and implementing agent architectures has been proposed in some recent studies. Specifically, the research in [56, 57] (Sabater et al., 2002 and Casali et al., 2008) propose breaking the

logical description of an agent into a set of contexts, each of which represents a different component of the architecture, and the interactions between these components are specified by means of bridge rules. A similar approach is followed in [58] (Dastani et al., 2007), where contextual deliberation of cognitive agents is achieved using a special extension of Defeasible Logic. On the other hand, in the multi-agent architectures proposed in [59, 60] (Cristani and Burato, 2009, Resconi and Kovalerchuk, 2009), context refers to a criterion, with respect to which an agent thinks it is important to evaluate an action. In our case, a context represents the viewpoint of each different agent in the system.

5.3 Contextual Defeasible Logic

The Contextual Defeasible Logic (CDL) is a language that provides a fully distributed approach for contextual reasoning in Ambient Intelligence environments. CDL is based on Defeasible Logic [47] which is skeptical, rule-based, and uses priorities to resolve conflicts among rules. Moreover, CDL adopts ideas from Multi-Context Systems (MCS) [35] which consist of a set of contexts and a set of inference rules (a.k.a. mapping rules) that enable information flow between different contexts. Essentially, the Multi-Context Systems model is enriched through defeasible rules, and priority relations that provide a preference ordering between system contexts regarding their reliability.

5.3.1 Representation Model

In CDL a Multi-Context System C is modeled as a collection of distributed context theories C_i: A context is defined as a tuple of the form(V_i, R_i, T_i) where V_i is the vocabulary used by C_i, R_i is a set of rules, and T_i is a preference relation on C.

V_i is a set of positive and negative literals. If q_i is a literal in V_i, $\sim q_i$ denotes the complementary literal, which is also in V_i. If q_i is a positive literal p then $\sim q_i$ is $\neg p$; and if q_i is $\neg p$, then $\sim q_i$ is p. We assume that each context uses a distinct vocabulary.

R_i consists of two sets of rules: the set of local rules and the set of mapping rules. The body of local rules is a conjunction of local literals (literals that are

contained V_i), while their head contains a single local literal. There are two types of local rules:

- Strict rules, of the form: $r_i^l: a_i^1, a_i^2, \dots a_i^{n-1} \rightarrow a_i^n$. They express local knowledge and are interpreted in the classical sense: whenever the literals in the body of the rule are strict consequences of the local theory, then so is the literal in the head of the rule. Local rules with empty body denote factual knowledge.
- Defeasible rules, of the form: $r_i^d: b_i^1, b_i^2, \dots b_i^{n-1} \Longrightarrow b_i^n$. They are used to express uncertainty, in the sense that a defeasible rule cannot be applied to support its conclusion if there is adequate contrary evidence.

Mapping rules associate local literals with literals from the vocabularies of other contexts (foreign literals). The body of each such rule is a conjunction of local and foreign literals, while its head contains a single local literal: $r_i^m: a_i^1, a_j^2, \dots a_k^{n-1} \Longrightarrow a_i^n$. r_i^m associates local literals of C_i (e.g. a_i^1) with local literals of C_j (a_j^2), C_i(a_k^{n-1}) and possibly other contexts. a_i^n is a local literal of the theory that has defined r_i^m (C_i).

Finally, each context C_i defines a trust level order Ti which expresses its confidence (trust) in the knowledge it imports from other contexts. More details about the reasoning model of CDL and how it has already been applied in Ambient Intelligence and Mobile Computing are available in [31, 32, 33, 36, 37].

A Distributed Environment Authorization Language: DEAL

In this chapter we present the formal high level logic-based language *DEAL (Distributed Environment Authorization Language)* for expressing authorization policies in distributed ambient environments. DEAL is based on Defeasible Logic and is able to support all the desirable features that were described in chapter 3. Moreover the implementation of the language as an extension to CDL enables DEAL to specify distributed authorizations by adopting the connection-based approach (described on chapter 4). Firstly, we describe the syntax of the language in detail. Secondly, we provide the semantics of the language through transformation to Defeasible Logic. Thirdly, we illustrate the appropriate extensions to the basic CDL algorithm in order to support DEAL policies. Finally, we present the implementation of the motivating scenarios which were discussed in chapter 1 using DEAL policies.

6.1 Language Syntax

In this section we provide the syntax of the language in detail by presenting the DEAL alphabet, rules and the expressive characteristics of the language through examples.

6.1.1 Alphabet of DEAL Language

The alphabet of DEAL language consists of four sets, the *constants* (C), the *variables* (V), the *predicate symbols* (P) and the *logical symbols* (L).

Constants and Variables: In DEAL policies constants and variables are used in the classical sense. More specifically, a constant has a specific nonchanging value while a variable has a changing value that varies. Constant symbols start with a lowercase letter while variable symbols start with an uppercase letter. A constant represents an environment element such as an AmI entity, a provided service (by an AmI entity), a resource or an action that is performed to a resource. On the other hand, a variable ranges over the constant set C. Thus, a variable represents any environment element which can be expressed by a constant. Moreover, a variable is instantiated when it is assigned a concrete value.

Predicate Symbols: Predicate symbols are symbols used to denote *predicate knowledge*. Predicate knowledge expresses some relation or some property of the environment elements. The value of a predicate symbol can be either *true* or *false*. This value denotes if the predicate knowledge can be verified based on a given knowledge base. Predicates have zero or more arguments (in order to express the environment elements) which are enclosed in parenthesis and are comma separated. These arguments may be constants, variables or other predicates. An unary predicate (with one argument) denotes a property while an n-arity predicate (n >1) denotes a relation. A predicate can be evaluated (assigned a value) only when all variables are instantiated (*ground* predicate). The predicate symbols of DEAL alphabet are described below in detail.

- *belong(X, Y)*: Predicate symbol *belong* represents that an element *X* belongs to a category of elements *Y*. Moreover, it may represent that a subcategory *X* belongs to an element category *Y*. The *X* element can either be an AmI entity, an AmI service, an action which is performed to a resource or an AmI resource.
- *right(X, Y)*: Predicate symbol *right* represents the privilege to perform an action *X* to an AmI resource *Y*. Moreover, the arguments *X, Y* may also represent a category of actions and a category of resources respectively.
- *grant(X, Y, Z):* Predicate symbol *grant* represents a positive authorization (permission) that is given by a grantor AmI entity *X* to a grantee AmI entity *Y* for consuming an AmI service *Z*. Moreover, the arguments *Y, Z* may also represent a category of AmI entities and a category of services respectively. Furthermore, the argument *Z* may also be a *right* predicate.
- *granted(Y, Z):* Predicate symbol *granted* represents a positive authorization (permission) in the exact same sense as it is specified for predicate *grant*. The only difference is that the grantor argument is omitted as it is assumed to be the local system.
- *superior(X, Y):* Predicate symbol *superior* represents a preference between a pair of conflicting rules (*X, Y*). More specifically, the rule with name denoted in *X* is preferred than the rule with name denoted in *Y*.

Apart from the above predicate symbols the user is able to define any predicate of *n*-arity in order to represent knowledge for a particular application domain. For example, if a business application for a particular company requires the specification of the property "*manager*" in order to represent persons that are project managers, the user is able to define the application dependent predicate *isManager(X)*. The predicate *isManager(X)* denotes the fact that person *X* is a manager.

A predicate symbol or its negation is defined as *literal* in DEAL language. The specification of a predicate negation and its semantics are explained in the description of DEAL logical symbols which is provided below.

Logical Symbols: The DEAL language supports the following logical symbols:

- *Strong Negation*: DEAL supports strong negation (a.k.a. classical negation) with the use of ¬ symbol in front of language predicates. Strong negation that is used in front of a predicate denotes contradictory knowledge from what the predicate expresses. In other words, given a data element *p* which is a grounded predicate then ¬ *p* represents the contradictory data element. Thus, positive truth values for both *p*, ¬ *p* leads to a knowledge conflict instance. Moreover, strong negations in front of predicates grant and *granted* expresses negative authorization (described in section 3.1) instead of positive. Therefore, strong negation enables the specification of the authorization conflict which is the most common type of knowledge conflict in authorization applications. Furthermore, strong negation can be used in front of any user-defined predicate in order to represent contradictory information. Therefore, it enables the specification of other user-defined types of knowledge conflicts.
- *Weak Negation*: DEAL supports weak negation (a.k.a. negation as failure) with the use of *not* keyword in front of language predicates. Weak negation that is used in front of a predicate denotes the absence of the predicate as a data element. In other words, given a data element *p* which is a grounded predicate, then *not p* is true if *p* is false (absent). In this

case, the contradictory data element $\neg p$ may or may not be true. The difference between strong and weak negation for a predicate *p* is that the first one represents the existence of negative (contradictory) information ($\neg p$) while the second one represents the absence of positive information (*p*). The two definitions are not equivalent in an environment where *p* and $\neg p$ may coexist. It should be noted also that weak negation is a common feature of nonmonotonic reasoning models.

- *Conjunction*: Deal supports logical conjunction of literals with the use of comma symbol.
- *Strict entailment*: Deal supports strict entailment with the use of $\leftarrow$ symbol. Strict entailment is used to express authorization rules with the classical sense of logical implication (deductive reasoning): Given a conjunction of literals *X* at the right side of the operator and a literal *Y* at the left, whenever *X* is true, *Y* can be derived as a logical consequence. The only restriction is that literal *Y* cannot be a weak negated predicate. A rule that is specified with strict entailment is called *strict rule*.
- *Defeasible entailment*: Deal supports defeasible entailment with the use of $\Leftarrow$ symbol. Defeasible entailment is used to express authorization rules in the following sense (defeasible reasoning based on rule preference): Given a conjunction of literals *X* at the right side of the operator and a literal *Y* at the left, whenever *X* is true, *Y* can be derived as a logical consequence, only if $\neg Y$ cannot be derived by a non-inferior (superior or neutral, described in section 2.4) conflicting rule. A rule that is specified with defeasible entailment is called *defeasible rule*. Defeasible entailment is also a common feature in nonmonotonic reasoning models.

6.1.2 Rules of DEAL Language

In DEAL language *direct knowledge* (a.k.a. *facts*) is expressed by rules with empty body. On the other hand, *derived knowledge* is expressed as conclusions of rules with non-empty body.

Moreover, in DEAL we distinguish three types of authorization rules, the *final rule*, the *priority rule* and the *hierarchy rule*. A DEAL rule may have the form of one of these types or it may be an authorization rule of another form specified by the

predicate and logical symbols of the language. The definitions of these rule types are described below in detail.

Definition 1. A hierarchy rule is a rule of the following form:

$$belong(X, Y) \quad \leftarrow \quad .$$

The hierarchy rule is used to support the representation of hierarchical categories. The hierarchy rule concludes to a transitive relation. For example, consider that it is given the direct knowledge of the following rules:

$$belong(a, b) \quad \leftarrow \quad .$$
$$belong(b, c) \quad \leftarrow \quad .$$

In this case, we conclude: $belong(a, c) \quad \leftarrow \quad .$

Definition 2. A final rule is a rule of the following form:

$$<Rule\_name> \; G \quad \Leftarrow \quad L_1, L_2, ..., L_n \quad .$$

The final rule concludes to literal G which represents a predicate of the set {*granted, grant*} or their respective strong negations {$\neg$ *granted,* $\neg$ *grant*} while L_1, L_2, ..., L_n is a conjunction of any literals that are supported in DEAL language. In other words, the final rule concludes to an authorization specified in G while its fulfillment requirements are specified in the conjunction of literals L_1, L_2, ..., L_n. Moreover, in case where the authorization (which is specified by literal G) refers to a category (as it is specified in predicate symbol description of section 6.1.1) then the authorization is inherited to every element of the category. Note, that categories may refer either to AmI entities, AmI services, actions (which are performed on AmI resources) or AmI resources. For example, consider the following rules encoding direct knowledge, where the first rule represents a category of AmI entities and the second rule specifies an authorization on the entity category:

$$belong(a, b) \quad \leftarrow \quad .$$
$$granted(b, q) \quad \Leftarrow \quad .$$

In this case, we conclude: $granted(a, q) \quad \Leftarrow \quad .$

Thus, a final rule supports hierarchical category authorization (described in section 3.3) that enables the authorizations which are specified to a category to be inherited to every category element.

Moreover, note that a final rule is specified with defeasible entailment. This is due to the fact that an authorization policy may include many and possibly contradictory final rules that lead to authorization conflicts. Thus, a final rule is specified as a defeasible rule which implies defeasible reasoning based on rule preference that is able to handle possible rule conflicts arising from contradictory rules.

Furthermore, note the <*Rule_name*> label in front of the final rule. DEAL enables the use of rule labels in front of any rule in order to specify its unique name. Rule labels are required for the specification of rule preferences.

Definition 3. A priority rule is a rule of the following form:

$$superior(X,\ Y) \quad \leftarrow \quad .$$

The priority rule is used to support the rule priorities feature (described in section 3.2) which is able to express a rule preference for resolving rule conflicts. Given a conflicting pair of rules (X, Y) where X, Y are the rule names specified in rule labels, it is declared that rule with name X is preferred to rule with name Y. Thus, rule with name X is the superior rule and rule with name Y is the inferior rule regarding rule preference. In case of a potential knowledge conflict from a contradictory pair of rules which are neutral regarding rule preference (not associated with any priority rule), both their conclusions are blocked (noRule-preference approach). In this way, inconsistency caused by contradictory conclusions is avoided.

The priority rule concludes to an acyclic relation. For example the direct knowledge of the following rules is considered invalid:

$$superior(a,\ b) \quad \leftarrow \quad .$$
$$superior(b,\ c) \quad \leftarrow \quad .$$
$$superior(c,\ a) \quad \leftarrow \quad .$$

6.1.3 Characteristics of DEAL Language

DEAL language has expressive characteristics that meet all the desirable features of a powerful authorization language for Ambient Intelligence environments which were discussed in chapter 3. We describe in detail how DEAL is able to support all of these features through examples that illustrate the expressive power of the language.

Firstly, the negative authorization feature is supported in DEAL by using the strong negation symbol ¬ in front of the authorization predicates *granted* and *grant* (as mentioned earlier in section 6.1.1). We describe below two examples which require negative authorizations and we provide their authorization policies using DEAL language.

Example 6.1. In an ambient classroom system, a student is not granted to read exam solutions if his teacher does not allow him to do so. In this example, in order to conclude negative authorization for a student we require a negative authorization from his teacher. The authorization policy for this example is specified below in DEAL language.

$$\neg\, granted(X, readSolutions) \Leftarrow isStudent(X),\ teaches(Y, X),\ \neg\, grant(Y, X, readSolutions).$$

Example 6.2. This example refers to the AmI scenario that was described in section 3.1. The authorization policy of the scenario was depicted in figure 3.1. This policy is specified below in DEAL language.

$$\neg\, grant('Antoniou', 'Bob', passExam) \Leftarrow .$$
$$\neg\, grant('Antoniou', 'Alice', passExam) \Leftarrow .$$
$$\neg\, grant('Antoniou', X, passExam) \Leftarrow isStudent(X),\ exceededTime(X).$$

Secondly, rule priorities is supported in DEAL by using rule labels in front of defeasible rules and the priority rule to denote rule preference. We describe below two

examples which require rule priorities and we provide their authorization policies using DEAL language.

Example 6.3. According to the access control system of a company, no one is allowed to access company money except for the accountants. The authorization policy can be represented with the following statements that are listed below.

1. *No one is allowed access to company money.*
2. *If a person is an accountant, he can access company money.*
3. *Statement 2 is preferred from statement 1.*

This policy is specified below in DEAL language.

<***label1***>	¬ *granted(X, accessMoney)*	⇐	.
<***label2***>	*granted(X, accessMoney)*	⇐	*isAccountant(X).*
<***label3***>	*superior(label2, label1)*	←	.

Example 6.4. This example refers to the AmI scenario that was described in section 3.2. The authorization policy of the scenario was depicted in figure 3.3. This policy is specified below in DEAL language.

<***label1***>	*grant('Smith', X, access('PA'))*	⇐	*isManager(X).*
<***label2***>	¬ *grant('Smith', X, access('PA'))*	⇐	*isEmployee(X), fired(X).*
<***label3***>	*grant('Smith', X, access('PA'))*	⇐	*isSubmanager(X).*
<***label4***>	*grant('Smith', X, access('PA'))*	⇐	*isAdministrator(X).*
<***label5***>	*superior(label1, label2)*	←	.
<***label6***>	*superior(label2, label3)*	←	.
<***label7***>	*superior(label2, label4)*	←	.

Thirdly, hierarchical categories can be represented in DEAL by using the hierarchy rule. The hierarchical categories authorization feature is supported in DEAL by using the authorization rule in conjunction with the hierarchy rule. We describe below four examples which require hierarchical category authorization and we provide their authorization policies using DEAL language.

Example 6.5. In a firewall system all IPs that belong to the category "*malicious*" are not granted the ftp service. The IPs that belong to the category "*malicious*" are "*ipA*" and "*ipB*". This example involves hierarchical categories of entities. The authorization policy for this example is specified below in DEAL language.

belong(ipA, malicious) ← .
belong(ipB, malicious) ← .
¬ *granted(malicious, ftpService)* ⇐ .

Example 6.6. A weather observation system allows access to weather forecast only to specific official sites such as "*weather.com*" and "*travelling.com*". The wind-information service belongs to the weather-forecast service . Moreover, the wind-direction and wind-strength services belong to the wind-information service. This example involves hierarchical categories of services. The authorization policy for this example is specified below in DEAL language.

belong(temperatureInformation, weatherForecast) ← .
belong(windInformation, weatherForecast) ← .
belong(windDirection, windInformation) ← .
belong(windStrength, windInformation) ← .
granted(site(weather.com), weatherForecast) ⇐ .
granted(site(travelling.com), weatherForecast) ⇐ .

Example 6.7. The administrator of a site is allowed any action, such as read or write, to the file that contains the users passwords. This example involves hierarchical categories of actions. The authorization policy for this example is specified below in DEAL language.

belong(read, fileAction) ← .
belong(write, fileAction) ← .
granted(admin, right(fileAction, userPasswords.txt)) ⇐ .

Example 6.8. The administrator of a site is allowed access to any of the user files. The category "*userPhotos*" belongs to "*userFiles*" category. The category "*userFiles*"

includes also profile information described in the file "*profile.txt*". Moreover, the photos that belong to "*userPhotos*" category are "*photoA.jpg* " and "*photoB.jpg*". This example involves hierarchical categories of objects. The authorization policy for this example is specified below in DEAL language.

belong(profile.txt, userFiles) ← .
belong(userPhotos, userFiles) ← .
belong(photoA.jpg, userPhotos) ← .
belong(photoB.jpg, userPhotos) ← .
granted(admin, right(access, userFiles)) ⇐ .

Finally, DEAL enables nonmonotonic reasoning with authorization policies. Firstly, nonmonotonic policies are supported by the use of weak negation. Specifically, if weak negation is used in front of a predicate *p* it implicitly represents a nonmonotonic rule that derives *not p* from the absence of *p* (failure to derive *p*) which means that future information about *p* may withdraw the previous conclusion. Thus, a rule that uses weak negation in its conditions inherits this nonmonotonicity. Secondly nonmonotonic policies are supported by the use of defeasible entailment. Defeasible entailment is a nonmonotonic feature because it enables the specification of defeasible rules which provide conclusions that in the presence of future information (from contradictory rules) may be withdrawn. The study in [64] provides more information about defeasible reasoning as a form of nonmonotonic reasoning. A nonmonotonic reasoning policy specified with defeasible rules was provided in example 6.4. We describe below two examples of nonmonotonic reasoning specified with weak negation and we provide their authorization policies using DEAL language.

Example 6.9. According to the access control system of a cinema, a person cannot enter the cinema if he has not bought a ticket. The authorization policy for this example is specified below in DEAL language.

¬ *granted(X, entry(cinema))* ⇐ *not hasBought(X, ticket).*

In the above example, we conclude negative authorization for a person (about entering the cinema), it there is no information that he has bought a

ticket. Future information about a ticket sell may withdrawn previous negative authorizations. Note the difference between strong negation of example 6.1. In the example 6.1, we conclude a negative authorization if there is knowledge of negative authorization from the teacher ($\neg$ *grant*) while in this example, we conclude a negative authorization if there is no knowledge about buying a ticket (*hasBought*).

Example 6.10. This example refers to the AmI scenario that was described in section 3.4. The authorization policy of the scenario was depicted in figure 3.8. This policy is specified below in DEAL language.

grant('Nick', X, right(accessPhoto, Y)) ← *not sold(Y).*
$\neg$ *grant('Nick', X, right(accessPhoto, Y))* ← *sold(Y).*

6.2 Language Semantics

DEAL language can be fully implemented in Defeasible Logic (described in chapter 5). In this section we illustrate the semantics of DEAL language through transformation to Defeasible Logic. More specifically, we describe in detail how the alphabet and rules of DEAL can be transformed to Defeasible Logic.

6.2.1 DEAL alphabet transformation

In this section we describe how the alphabet of DEAL can be fully represented by Defeasible Logic (DL). More specifically, we illustrate how the constants, the variables, the predicate and logical symbols of DEAL are specified in DL.

Firstly, DL supports constants and variables exactly as they are specified in DEAL. Secondly, DL enables the specification of user defined predicates. Thus, all DEAL predicates can be represented in DL. Finally, DL supports directly the logical symbols of strong negation, logical conjunction, strict and defeasible entailment as they are defined in DEAL. The only logical symbol that is not provided directly in DL is weak negation. However, weak negation can be specified as a predicate symbol indirectly by DL elements. More specifically, every declaration of *not X* where *X* is a language literal can be equivalently replaced with *not(X)* (where *not* is a predicate and *X* is an argument) and the addition of the following rules.

$$not(X) \Longleftarrow .$$
$$\neg\, not(X) \Longleftarrow X.$$

The first defeasible rule expresses that the absence of X (which is specified by predicate *not*) is always concluded. In other words, the weak negation of *X* which is defined as *not(X) is* always derived defeasibly (as conclusion of a defeasible rule). The second rule is used to block the conclusion of the first when *X* is present as a data element.

6.2.2 DEAL rules transformation

In this section we describe how the semantics of DEAL rules can be fully expressed by Defeasible Logic. More specifically, we illustrate how the semantics of the priority, the hierarchy and the final rule are specified in DL.

Firstly, we describe how the priority rule is expressed in Defeasible Logic. The priority rule is associated with rule names which (in DEAL) are specified in rule labels in front of rule definitions. Defeasible Logic expresses rule names also in front of rule definitions. An example of a DL rule with name *r1* is provided below.

$$r1: \quad bird(X) \leftarrow flies(X).$$

The priority rule in DEAL is used to support the rule priorities feature (described in section 3.2) that enables the specification of a rule preference for conflict resolution. The rule priorities feature is expressed in DL with the support of superiority relations. A superiority relation is defined with the use of $>$ logical symbol. More specifically, given a rule name *R1* at the left side of the operator and a rule name *R2* of a conflicting rule at the right side, it is denoted that *R1* is preferred to *R2*. Moreover, the superiority relation is an acyclic relation on R (the set of rules in the theory).Thus, the priority rule is specified as a superiority relation fact in DL.

Moreover, in case of a potential knowledge conflict from a contradictory pair of DL rules which are neutral regarding rule preference (not associated with any superiority relation fact), both their conclusions are blocked (noRule-preference approach). In this way, inconsistency caused by contradictory conclusions is avoided.

Secondly, we describe how the hierarchy rule is expressed in Defeasible

Logic. The hierarchy rule is used in DEAL to specify the hierarchical categories feature. Moreover, the hierarchy rule concludes to a transitive relation. In Defeasible Logic the hierarchy rule is specified exactly as in DEAL while the transitivity of the *belong* relation is supported with the following rule.

belong(X, Y) ← *belong(X, Z), belong(Z, Y).*

The only requirement is that the information expressed by the *belong* predicate is acyclic. For instance, the cycle expressed by the facts *belong(a, b), belong(b, c)* and *belong(c, a)* is prohibited, as a query of the form *belong(a, c)?* would cause an infinite loop. In case of cycles, in the information which is specified by the *belong* predicate, the hierarchy rule can be alternatively expressed in DL by the following two rules. The additional predicate *belongTo* is used to avoid the infinite loops. In this case the query *belong(a, c)?* which is evaluated over the facts *belongTo(a, b), belongTo(b, c)* and *belongTo(c, a)* is evaluated to true.

belong(X, Y) ← *belongTo(X, Y).*
belong(X, Y) ← *belongTo(X, Z), belong(Z, Y).*

Thirdly, we describe how the final authorization rule is expressed in Defeasible Logic. Final rules are used in DEAL to specify negative and positive authorizations under different circumstances. Moreover, the final rule is associated with the hierarchical category authorization feature (described in section 3.3) that enables the authorizations which are specified to a category to be inherited to every category element.

In Defeasible Logic the final rule is specified exactly as in DEAL while the hierarchical category authorization feature is supported with the following rules. As described in section 3.3 hierarchical categories can be specified for grantee entities, services, actions and objects.

Firstly, the rules given below specify derived authorizations among grantee entity hierarchies in Defeasible Logic.

granted(X, Q) ⇐ *belongTo(X, Y), granted(Y, Q).*
¬ granted(X, Q) ⇐ *belongTo(X, Y), ¬ granted(Y, Q).*
grant(G, X, Q) ⇐ *belongTo(X, Y), grant(G, Y, Q).*
¬ grant(G, X, Q) ⇐ *belongTo(X, Y), ¬ grant(G, Y, Q).*

Secondly, the following rules specify derived authorizations among service hierarchies in Defeasible Logic.

granted(X, Q) ⇐ *belongTo(Q, Y), granted(X, Y).*
¬ *granted(X, Q)* ⇐ *belongTo(Q, Y),* ¬ *granted(X, Y).*
grant(G, X, Q) ⇐ *belongTo(Q, Y), grant(G, X, Y).*
¬ *grant(G, X, Q)* ⇐ *belongTo(Q, Y),* ¬ *grant(G, X, Y).*

Thirdly, the following rules specify derived authorizations among action hierarchies in Defeasible Logic.

granted(X, right(A,O)) ⇐ *belongTo(A, Y), granted(X, right(Y,O)).*
¬ *granted(X, right(A,O))* ⇐ *belongTo(A, Y),* ¬ *granted(X, right(Y,O)).*
grant(G, X, right(A,O)) ⇐ *belongTo(A, Y), grant(G, X, right(Y,O)).*
¬ *grant(G, X, right(A,O))* ⇐ *belongTo(A, Y),* ¬ *grant(G, X, right(Y,O)).*

Finally, the following rules specify derived authorizations among object hierarchies in Defeasible Logic.

granted(X, right(A,O)) ⇐ *belongTo(O, Y), granted(X, right(A,Y)).*
¬ *granted(X, right(A,O))* ⇐ *belongTo(O, Y),* ¬ *granted(X, right(A,Y)).*
grant(G, X, right(A,O)) ⇐ *belongTo(O, Y), grant(G, X, right(A,Y)).*
¬ *grant(G, X, right(A,O))* ⇐ *belongTo(O, Y),* ¬ *grant(G, X, right(A,Y)).*

In this section, we illustrated how the DEAL rules can be expressed in Defeasible Logic. In other words, we provided the semantics of DEAL rules specified in Defeasible Logic.

In conclusion, we described how DEAL language can be fully implemented in Defeasible Logic. The transformation of DEAL language to Defeasible Logic provides formal semantics for the language. Moreover, in the propositional case Defeasible Logic has linear complexity as described in [65]. Therefore, DEAL language which can be fully transformed to Defeasible Logic has also linear complexity in the propositional case.

6.3 Contextual Defeasible Logic Extensions

Contextual Defeasible Logic (CDL) is a language that enables reasoning about context in ambient intelligence environments. In this section we provide the algorithm extensions to the basic reasoning processing of CDL in order to support DEAL policies.

As described in chapter 5, CDL models a multi-context system P as a collection of distributed local rule theories Pi in a P2P system: P={Pi}, i=1,…,n. Each system node (context) has a proper distinct vocabulary Vi defined as a set of literals and their negations, and a unique identifier i. Each local theory consists of a set of rules that contain only literals from the local vocabulary. These rules are interpreted in the classical sense: whenever the literals in the body of a local rule are consequences of the local theory, then so is the conclusion of the rule Each node also defines a set of mappings that associate literals from its own vocabulary (local literals) with literals from the vocabulary of other peers (foreign literals). Mappings are modeled as defeasible rules which can be defeated in the existence of adequate contrary evidence. Their conclusions are labeled by local literals. Finally, each node Pi defines a preference order Ti, which includes a subset of the system nodes, and expresses the trust that Pi has in the other system nodes.

Contextual reasoning proceeds roughly as follows: when a peer P processes a query q, it may query through bridge rules other peers, which in turn may pass on queries to further peers. Based on the information that is collected, P builds a support set and a blocking set for the query q. The support and blocking set contain information about the peers that provide respectively, positive and negative responses. A comparison between the sets is achieved, based on the trust that P places to other peers. Finally, the comparison result leads to a positive or negative conclusion. In the simplest case, a peer Q responses to a query issued by peer P only with true/false. In more complex strategies [37], Q passes on further information regarding the support and blocking sets that determine the answer it returns to P. The selection of the appropriate strategy for an AmI system depends, among others, on the requirements regarding efficiency and privacy protection.

The assumption made in CDL processing is that a requester is always granted to consume any service and the CDL algorithm only tries to provide an answer to the requester. In other words, the current implementation of CDL always proceeds in

concluding and returning an answer for an issued query without any authorization restrictions. However, there are cases where the system should not proceed on answering the request due to privacy and security reasons. Real applications require an authorization policy that specifies when a requester should be provided or denied a specific service. The limitation of CDL is that it does not deal with privacy and security issues as it does not support any kind of authorization control.

The extensions of CDL aim on addressing authorization issues of reasoning in Multi-Context systems. The CDL approach is enriched in order to support DEAL authorization policies. The extended version of CDL deals with the following problem: *Given a MCS C and a query about literal p_i issued by context C_0 to context C_i , if the request pair (C_0, p_i) is an authorized pair, compute the truth value of p_i*. The C_i context triggers the query "*granted*(C_0, p_i)*?*" in order to determine whether the request pair is authorized. The answer to the query is determined by C_i context theory that includes a DEAL authorization policy. The authorization policy may involve multiple authorization rules that finally conclude to one of the following answers for the query:

(a) true: Indicating that the request pair is authorized.
(b) false: Indicating that the request pair is not authorized.
(c) undefined: Indicating that neither *true* nor *false* could be derived.

The *negative* (*false*) or *undefined* answer implies that context C_i should not provide context C_0 with an answer for the issued query p_i, due to various privacy and security reasons (e.g. C_0 context may be on a malicious requester list). In this case, C_i should reply to C_0 that the truth value of p_i is undefined. The *positive* (*true*) answer implies that context C_i should proceed on computing and replying the truth value of p_i to context C_0. In this case, C_i should call the *P2P_DR* distributed algorithm that implements the basic strategy of contextual reasoning in order to compute the truth value of p_i.

The *P2P_DEAL* algorithm provides the extended version of the CDL reasoning process. The pseudocode of the algorithm is presented below. The algorithm proceeds in five steps. In the first step, *P2P_DEAL* calls *Accept* that triggers the query "*granted*(C_0, p_i)*?*" on the context theory of C_i. Essentially, *Accept* calls *P2P_DR* algorithm in order to compute the truth value of *granted*(C_0, p_i) literal.

The query answer is returned in Ans_{C_o} variable. In the following three steps, *P2P_DEAL* checks variable Ans_{C_o} for a *negative* or *undefined* value. In this case, it assigns the answer *undefined* in variable Ans_{p_i} (that represents the truth value of p_i) and returns it. It also assigns, the empty set to the support (SS_{p_i}) and blocking (BS_{p_i}) set. Alternatively, if Ans_{C_o} contains a positive value (true) the algorithm proceeds in step five. In this case, *P2P_DEAL* calls *P2P_DR* algorithm in order to compute the truth value of p_i literal.

The difference of *P2P_DEAL* with the previous CDL algorithm is that the latter always call *P2P_DR* algorithm for query evaluation without implementing any form of authorization control. *P2P_DR* is an algorithm which is provided in the previous version of CDL. For completeness, we describe it also here, after the presentation of *P2P_DEAL* algorithm.

The input parameters of P2P_DEAL are:

- p_i: The queried literal.
- C_0: The context that issues the query.
- C_i: The context that receives the query.
- $Hist_{p_i}$: The list of pending queries ([p_1, ..., p_i])
- T_i: The preference ordering of C_i

The output parameters of *P2P_DEAL* are:

- SS_{p_i}: A set of foreign literals of C_i denoting the Supportive set of p_i.
- BS_{p_i}: A set of foreign literals of C_i denoting the Blocking set of p_i.
- Ans_{p_i}: The answer returned for p_i.

Below, we provide the pseudocode of P2P_DEAL algorithm.

P2P_DEAL ($p_i, C_0, C_i, Hist_{p_i}, T_i, SS_{p_i}, Ans_{p_i}$)

1. call Accept(C_0, C_i, p_i, Ans_{C_o})
2. **if** Ans_{C_o} = false **or** Ans_{C_o} = undefined

3. $Ans_{p_i} \leftarrow undefined, SS_{p_i} \leftarrow \emptyset, BS_{p_i} \leftarrow \emptyset$
4. terminate
5. call P2P_DR ($p_i, C_0, C_i, Hist_{p_i}, T_i, SS_{p_i}, Ans_{p_i}$)

P2P_DR is a distributed algorithm for query evaluation that deals with the following problem: *Given a MCS C, and a query about literal p_i issued to context C_i, compute the truth value of p_i*.Essentially, *P2P_DR* returns one of the following values for an arbitrary literal p_i:

(a) true: Indicating that p_i is a logical consequence of *C*.
(b) false: Indicating that p_i is not a logical consequence of *C*.
(c) undefined: Indicating that neither *true* nor *false* could be derived.

P2P_DR proceeds in four main steps. In the first step, *P2P_DR* determines whether p_i, or its negation $\sim p_i$ are consequences of the strict local rules of C_i, returning *true/false* respectively as an answer for p_i and terminates.

In the second step, *P2P_DR* calls *Support* to determine whether there are applicable and unblocked rules with head p_i. We call *applicable* those rules that for all literals in their body *P2P_DR* has computed *true* as their truth value, while *unblocked* are the rules that for all literals in their body *P2P_DR* has computed either *true* or *undefined* as their truth value. *Support* returns two data structures for p_i: (a) the Supportive Set of *pi* (SS_{p_i}), which is the set of foreign literals used in the most preferred (according to T_i) chain of applicable rules for p_i ; and (b) the Blocking Set of p_i (BS_{p_i}), which is the set of foreign literals used in the most preferred chain of unblocked rules for p_i (BS_{p_i}). If there is no unblocked rule for p_i ($BS_{p_i} = \emptyset$), the algorithm returns *false* as an answer and terminates.

In the third step, similarly to the second, *P2P_DR* calls *Support* to compute the respective constructs for $\sim p_i$ ($SS_{\sim p_i}, BS_{\sim p_i}$).

In the last step, *P2P_DR* uses the constructs computed in the previous steps and the preference order T_i, to determine the truth value of p_i. In case there is no unblocked rule for $\sim p_i$ ($BS_{\sim p_i} = \emptyset$), or SS_{p_i} is computed by *Stronger* to be *stronger* than $BS_{\sim p_i}$, *P2P_DR* returns *true* as an answer for p_i. That SS_{p_i} is *stronger* than BS_{p_i}

means that the chains of applicable rules for p_i involve information from contexts that are preferred by C_i to the contexts that are involved in the chain of unblocked rules for $\sim p_i$. In case there is at least one applicable rule for $\sim p_i$, and BS_{p_i} is *not stronger* than $SS_{\sim p_i}$, *P2P_DR* returns *false* as an answer for p_i. In any other case, the algorithm returns *undefined.*

The context that is called to evaluate the query for $p_i(C_i)$ returns through Ans_{p_i} the truth value of the literal it is queried about. SS_{p_i} and BS_{p_i} are returned to the querying context (C_0) only if the two contexts (the querying and the queried one) are actually the same context. Otherwise, the empty set is assigned to both $SS_{p_i} and\ BS_{p_i}$ and returned to C_0. In this way, the size of the messages exchanged between different contexts is kept small. $Hist_{p_i}$ is a structure used by Support to detect loops in the global knowledge base.

The input parameters of *P2P_DR* are:

- p_i: The queried literal.
- C_0: The context that issues the query.
- C_i: The context that receives the query.
- $Hist_{p_i}$: The list of pending queries ([p_1, ..., p_i])
- T_i: The preference ordering of C_i

The output parameters of *P2P_DR* are:

- SS_{p_i}: A set of foreign literals of C_i denoting the Supportive set of p_i.
- BS_{p_i}: A set of foreign literals of C_i denoting the Blocking set of p_i.
- Ans_{p_i}: The answer returned for p_i.

Below, we provide the pseudocode of *P2P_DR* algorithm.

P2P_DR ($p_i, C_0, C_i, Hist_{p_i}, T_i, SS_{p_i}, Ans_{p_i}$)

call local_alg ($p_i, localAns_{p_i}$)
if localAns = true **then**

$Ans_{p_i} \leftarrow true, SS_{p_i} \leftarrow \emptyset, BS_{p_i} \leftarrow \emptyset$

terminate

call local_alg ($\sim p_i, localAns_{p_i}$)

if $localAns_{\sim p_i} = true$ **then**

$Ans_{p_i} \leftarrow false, SS_{p_i} \leftarrow \emptyset, BS_{p_i} \leftarrow \emptyset$

terminate

call Support ($p_i, Hist_{p_i}, T_i, \sup_{p_i}, unb_{p_i}, SS_{p_i}, BS_{p_i}$)

if $unb_{p_i} = false$ **then**

$Ans_{p_i} \leftarrow false, SS_{p_i} \leftarrow \emptyset, BS_{p_i} \leftarrow \emptyset$

Terminate

$Hist_{\sim p_i} \leftarrow \left(Hist_{p_i} - \{ p_i \}\right) \cup \{ \sim p_i \}$

call Support ($\sim p_i, Hist_{\sim p_i}, T_i, \sup_{\sim p_i}, unb_{\sim p_i}, SS_{\sim p_i}, BS_{\sim p_i}$)

if

$\sup_{p_i} = true\ and\ (\ unb_{\sim p_i} = false\ or\ Stronger\left(SS_{p_i}, BS_{\sim p_i}, T_i\ \right) = SS_{p_i})$

then

$Ans_{p_i} \leftarrow true$

if $C_0 \neq C_i$ **then**

$SS_{p_i} \leftarrow \emptyset, \quad BS_{p_i} \leftarrow \emptyset$

else if $\sup_{\sim p_i} = true\ and\ Stronger\left(\ BS_{p_i}, SS_{\sim p_i}, T_i\right) \neq BS_{p_i}$ **then**

$Ans_{p_i} \leftarrow false, SS_{p_i} \leftarrow \emptyset, BS_{p_i} \leftarrow \emptyset$

else

$Ans_{p_i} \leftarrow undefined$

if $C_0 \neq C_i$ **then**

$SS_{p_i} \leftarrow \emptyset, BS_{p_i} \leftarrow \emptyset$

local_alg is called by *P2P_DR* to determine whether the truth value of the queried literal can be derived from the strict local rules of a context theory. We should note that, for sake of simplicity, we assume that there are no loops in the local context

theories. *local_alg* returns either *true* or *false* as a local answer for the queried literal.

The algorithm parameters are:

- p_i: The queried literal.
- $localAns_{p_i}$: The local answer for p_i (output)

Local_alg ($p_i, localAns_{p_i}$)

for all $r_i \in R^s[p_i]$ **do**
for all $b_i \in body(r_i)$ **do**
call local_alg($b_i, localAns_{b_i}$)
if for all b_i: $localAns_{b_i} = true$ **then**
return $localAns_{p_i} = true\ and\ terminate$
return $localAns_{p_i} = false$

Support is called by P2P_DR to determine whether there are applicable and unblocked rules for p_i. In case there is at least one applicable rule for p_i, *Support* returns $\text{sup}_{p_i} = true$; otherwise, it returns $\text{sup}_{p_i} = false$. Similarly, $unb_{p_i} = true$ is returned when there is at least one unblocked rule for p_i; otherwise, $unb_{p_i} = false$.

Support also returns two data structures for p_i:

- SS_{p_i}: $the\ Supportive\ Set\ for\ p_i$. This is a set of literals representing the most preferred (according to T_i) chain of applicable rules for p_i.
- BS_{p_i}: $the\ Blocking\ Set\ for\ p_i$. This is a set of literals representing the most preferred (according to T_i) chain of unblocked rules for p_i.

To compute these structures, *Support* checks the applicability of the rules with head p_i, using the truth values of the literals in their body, as these are evaluated by P2P_DR. To avoid loops, before calling P2P_DR, it checks if the same query has been issued before during the running call of P2P_DR. In this case, it marks the rule with a cycle value, and proceeds with the remaining body literals. For each applicable rule r_i. *Support* builds its Supportive Set, S_{r_i} ; this is the union of the set of foreign

literals contained in the body of r_i with the Supportive Sets of the local literals contained in the body of the rule. Similarly, for each unblocked rule r_i, it computes its Blocking Set BS_{r_i}using the Blocking Sets of its body literals. Support computes the Supportive Set of p_i, SS_{p_i} , as the strongest rule Supportive Set SS_{r_i} ; and its Blocking Set, BS_{p_i}, as the strongest rule Blocking Set BS_{r_i} , using the Stronger function.

The input parameters of *Support* are:

- p_i: The queried literal.
- $Hist_{p_i}$: The list of pending queries ([p_1, ..., p_i])
- T_i: The preference ordering of C_i

The output parameters of *Support* are:

- sup_{p_i} : which indicates whether p_i is supported in C.
- unb_{p_i} : which indicates whether p_i is unblocked in C.
- SS_{p_i}: A set of foreign literals of C_i denoting the Supportive set of p_i.
- BS_{p_i}: A set of foreign literals of C_i denoting the Blocking set of p_i.

Support ($p_i, Hist_{p_i}, T_i, \text{sup}_{p_i}, unb_{p_i}, SS_{p_i}, BS_{p_i}$)
$sup_{p_i} \leftarrow false$
$unb_{p_i} \leftarrow false$
For all $r_i \in R[p_i]$ **do**
$cycle(r_i) \leftarrow false$
$SS_{r_i} \leftarrow \emptyset$
$BS_{r_i} \leftarrow \emptyset$
For all $b_t \in body(r_i)$ **do**
if $b_t \in Hist_{p_i}$ **then**
$cycle(r_i) \leftarrow true$
$BS_{r_i} \leftarrow BS_{r_i} \cup \{d_t\}\{d_t \equiv b_t \; if \; b_t \notin V_i; otherwise$
$d_t is \; the \; first \; foreign \; literal \; of \; C_i \; added \; in \; Hist_{p_i} after \; b_t\}$
else
$Hist_{b_t} \leftarrow Hist_{p_i} \cup \{b_t\}$

call P2P_DR($b_t, C_i, C_t, Hist_{b_t}, T_t, SS_{b_t}, BS_{b_t}, Ans_{b_t}$)

if $Ans_{b_t} = false$ **then**

stop and check the next rule

else if $Ans_{b_t} = undefined\ or\ cycle(r_i) = true$ **then**

cycle(r_i) ←$true$

if $b_t \notin V_i$ **then**

$BS_{r_i} \leftarrow BS_{r_i} \cup \{ b_t\}$

else

$BS_{r_i} \leftarrow BS_{r_i} \cup BS_{b_t}$

else

if $b_t \notin V_i$ **then**

$BS_{r_i} \leftarrow BS_{r_i} \cup \{ b_t\}$

$SS_{r_i} \leftarrow SS_{r_i} \cup \{ b_t\}$

else

$BS_{r_i} \leftarrow BS_{r_i} \cup BS_{b_t}$

$SS_{r_i} \leftarrow SS_{r_i} \cup SS_{b_t}$

if $unb_{p_i} = false\ or\ Stronger(BS_{r_i}, BS_{p_i}, T_i) = BS_{r_i}$ **then**

$BS_{p_i} \leftarrow BS_{r_i}$

unb_{p_i} ←$true$

if $cycle(r_i) = false$ **then**

if $\sup_{p_i} = false\ or\ Stronger(SS_{r_i}, SS_{p_i}, T_i) = SS_{r_i}$ **then**

$SS_{p_i} \leftarrow SS_{r_i}$

$sup_{p_i} \leftarrow true$

The *Stronger*(A, B, T_i) function computes the *strongest* between two sets of literals, *A* and *B* according to the preference order T_i. A literal $a_k$is preferred to a literal b_l, if there is a list L_j in $T_i$such that C_k and C_l are both part of that list and C_k precedes C_l. It must be noted that literals might not be comparable if there is no list in T_i containing both contexts from which these literals are derived. So in case where literals of two sets are not comparable then stronger function returns *undecided* for which set is the stronger.

Stronger (A, B, T_i)

if $\exists b_l \in B : \forall a_k \in A, \exists L_j \in T_i : C_l, C_k \in L_j$ ***and*** C_k *precedes* C_l *in* L_j **then**

Stronger = A

else if $\exists a_k \in A : \forall b_l \in B, \exists L_j \in T_i : C_k, C_l \in L_j$ ***and*** C_l *precedes* C_k *in* L_j **then**

Stronger = B

else

Stronger = None

return *Stronger*

The current extensions of the CDL algorithm that implement authorization control do not change the order of complexity of the previous algorithm. Essentially, the P2P_DEAL algorithm calls twice the P2P_DR algorithm in steps 1,5 while the other steps include conditional and assignment statements that check and init the value of variables. Thus, it is obvious that the order of complexity depends on the P2P_DR algorithm which is the previous CDL algorithm. The complexity of P2P_DR algorithm is provided in the proposition 3 of [37] which is presented below.

Proposition: *The total number of calls of P2P_DR that are required for the evaluation of a single query is in the worst case* $O(n \ x \ \sum P(n,k))$*, where n stands for the total number of literals in the system,* $\sum$ *expresses the sum over k = 0, 1, . . . , n, and* $P(n,k)$ *stands for the number of permutations with length k of n elements. If each of the literals in the system is defined by a different context, then the total number of messages exchanged between the system contexts for the evaluation of a query is* $O(2 \ x \ n \ x \ \sum P(n,k))$.

We should note that $\sum P(n,k)$ stands between 2^n and $n!*2^n$. In addition, the complexity of P2P_DR algorithm in an acyclic multi-context system is provided in the proposition 4 of [37] which is presented below.

Proposition: *In acyclic MCS, the total number of calls of P2P_DR that are required for the evaluation of a single query is in the worst case* $O(c \ x \ n)$*, where c stands for the total number of contexts in the system, and n stands for the total number of literals in the system. If each of the literals in the system is defined by a different context, then the total number of messages exchanged between the system contexts for the evaluation of a query is* $O(2 \ x \ c \ x \ n)$.

A complexity analysis for each different CDL strategy that deals with contextual reasoning is provided in [37]. The study in [36] presented the initial experiences gained from the deployment of contextual defeasible reasoning in real environments and discusses performance and scalability issues of the approach.

In conclusion, the implementation of DEAL on top of CDL framework provides two main advantages. Firstly, the CDL algorithm is enriched with authorization control. Secondly, CDL enables DEAL to specify distributed authorizations based on the connection-based approach (described on chapter 4).

6.4 Motivating Scenarios Implementation

In this chapter we present technical information about the implementation of the motivating scenarios that were described in section 1.3. Moreover, we specify the authorization policies of the scenarios in DEAL language. The first scenario refers to an Ambient Intelligence hospital environment and focuses on the protection of medical data, while the second scenario refers to an Ambient Intelligence university and focuses on the access control of secretarial services.

6.4.1 Implementation of AmI Hospital Authorization Scenario

The implementation of this scenario took place at ICS-FORTH institute facilities. Moreover, the implementation involved the following AmI devices, three cell phones that belong to the three doctors Bob, Trudy and Alice and four desktop computers that belong to the management office, and the hospital departments of Cardiology, Xray and Gastroenterology respectively.

The implementation of the information flow (which is depicted in figure 1.2) was based on the wireless networks of ICS-FORTH institute while the communication technologies that were used are SMS (short message service) messages via the GSM cellular network and P2P connections based on Bluetooth. More specifically, these technologies were used as follows. The individuals of the scenario are able to send requests, with their cell phones, to the management office computer in the form of SMS messages. The management office computer responds to the requests via SMS messages. Moreover, the management office computer

communicates with the computers of the departments via P2P connections based on Bluetooth in order to receive additional information about patients.

Moreover, the computers of this scenario are equipped with the latest version of CDL application which is able to address authorization issues via DEAL language and some CDL algorithm extensions. As for the technical details of the CDL application, the CDL algorithm is implemented in Java language. Moreover, as a basic reasoning engine it is used a Java implementation of a Prolog engine which is called TuProlog (which is described in the appendix A.1). The reasoning engine is preloaded with a metaprogram (which is provided in the appendix A.2) that simulates the proof theoretic semantics of Defeasible Logic and another metaprogram (which is provided in appendix A.3) that simulates the proof theoretic semantics of DEAL language.

Furthermore, in the implementation of this scenario we make identification and authentication assumptions because we focus on the authorization problem. More specifically, we assume that the requesters are identified and their identities are verified successfully when an intelligent device receives a request.

As for the policy of the management office for answering a particular request, it is expressed below in CDL language. The rules demonstrate that a received query of the form "*readyResults(X,Y)*", where *X* is a specific patient and *Y* is a specific type of exams, requires additional information from the specified department in order to be answered. For example, a received query that asks if the cardiology exams of a patient *X* are ready (*readyResults(X,'Cardiology')*) requires an additional query (*readyCardioExams(X)*) to be sent and answered by the computer of the Cardiology department.

r_1^m: *readyResults(X, 'Cardiology')* $\Leftarrow$ *readyCardioExams(X)*$_{CardioDep}$.

r_2^m: *readyResults(X, '*Xray*')* $\Leftarrow$ *readyXrayExams(X)*$_{\text{XrayDep}}$.

r_3^m: *readyResults(X, '*Gastroenterology*')* $\Leftarrow$ *readyGastroExams(X)*$_{\text{GastroDep}}$.

Moreover, the policy of the management office for authorizing a particular request is expressed below in DEAL language. The following rules of the authorization policy correspond to the authorization statements which are described in section 1.3.1.

<deal1> granted(X, readyResults(Y, Z)) ⇐ doctor(X), treat(X, Y).

<deal2> granted(X, readyResults(Y, Z)) ⇐ trainee(X),
grant(W, X, readyResults(Y, Z))
doctor(W), treat(W,Y).

<deal3> ¬ granted(X, readyResults(Y, Z)) ⇐ doctor(X), retired(X).

<deal4> superior(deal3, deal1) ← .

<deal5> belong(X,'Doctors') ← doctor(X).

<deal6> belong(X,'Doctors') ← trainee(X).

<deal7> granted('Doctors', diseaseOutbreak(X)) ⇐ .

<deal8> granted('Doctors', incidentsAbove(X,Y)) ⇐ .

As for the knowledge base of the management office, it is provided below in DEAL language.

doctor('Bob') ← .

treat('Bob', 'Mary') ← .

treat('Bob', 'George') ← .

trainee('Alice') ← .

grant(Bob', 'Alice', readyResults('George', Z)) ← .

doctor('Trudy') ← .

treat('Trudy', 'George') ← .

retired('Trudy') ← .

diseaseOutbreak('H1N1') ← .

Moreover, the implementation of this scenario involved different types of requests to the management office computer by the cell phones of the individuals. In addition, at the time when the requests were performed the computers of the hospital departments contained the knowledge which is specified below in CDL language.

Firstly, the Cardiology department contained the following knowledge:

readyCardioExams('Mary') ← .

Secondly, the Xray department contained the following knowledge:

readyXrayExams('George') ← .

Thirdly, the Gastroenterology department contained the following knowledge:

readyGastroExams('George') ← .

Furthermore, in this scenario there were preformed several requests. The replies to these requests verify that the system implements authorization control based on its authorization policy. To be more specific, we describe in detail the requests and the replies of the system.

First of all, we performed the query *readyResults('Mary', 'Cardiology')* by Bob's cell phone to the management office. The management office authorization policy concluded that Bob is authorized for this request due to rule *deal1* and its knowledge base. Thus, the system proceeded on processing and answering the request. The particular request is associated with the CDL mapping rule *r1* for distributed reasoning. Based on this rule, the additional request *readyCardioExams('Mary')* is forwarded to the Cardiology department in order to be answered. The Cardiology department answered with *'yes'* due to its knowledge base, thus the rule *r1* concludes the value *true* for the query *readyResults('Mary', 'Cardiology')*. Therefore the management office replies to Bob the answer 'yes'.

Additionally, we performed the query *diseaseOutbreak('H1N1')* by Bob's cell phone to the management office. The management office authorization policy concluded that Bob is authorized for this request due to rules *deal5, deal7* and its knowledge base. Thus, the system proceeded on processing and answering the request. The particular request is not associated with any CDL mapping rule for distributed reasoning but there is local knowledge on the management office for concluding the value *true* for the query. Thus, the management office replies to Bob the answer *'yes'*.

Moreover, we performed the queries *readyResults('George', 'Xray')* and *readyResults('George', 'Gastroenterology')* by Alice's cell phone to the management office. The management office authorization policy concluded that Alice is authorized for these requests due to rule *deal2* and its knowledge base. Thus, the system proceeded on processing and answering the requests. The particular requests are associated with the CDL mapping rules *r2* and *r3* for distributed reasoning. Based on these rules, the additional requests *readyXrayExams('George')* and

readyGastroExams('George') are forwarded to the Xray and Gastroenterology department respectively, in order to be answered. The Xray and Gastroenterology departments answered with *'yes'* due to their knowledge base. Thus, the rule *r2* concludes the value *true* for the query *readyResults('George', 'Xray')* and the rule *r3* concludes the value *true* for the query *readyResults('George', 'Gastroenterology')*. Therefore the management office replies to Alice the answer *'yes'* for both queries.

In addition, we performed the query *incidentsAbove('H1N1', 4)* by Alice's cell phone to the management office. The management office authorization policy concluded that Alice is authorized for this request due to rules *deal6, deal8* and its knowledge base. Thus, the system proceeded on processing and answering the request. The particular request is not associated with any CDL mapping rule for distributed reasoning and there is not any local knowledge on the management office for concluding the value *true* for the query. Thus, the management office replies to Alice the answer *'no'*.

Finally, we performed the query *readyResults('George', 'Xray')* by Trudy's cell phone to the management office. The management office authorization policy concluded that Trudy is not authorized for this request due to rules *deal1*, *deal3* and *deal4* and its knowledge base. Thus, the system did not proceed on processing and answering the request. Therefore, the management office replies to Trudy the answer *'undefined'*.

Overall, in the implementation of this scenario, the response of the management office system is correct based on its authorization policy.

6.4.2 Implementation of AmI University Authorization Scenario

The implementation of this scenario took place at ICS-FORTH institute facilities. Moreover, the implementation involved the following AmI devices, five cell phones that belong to the individuals Bob, Trudy, Alice, Mr. Antoniou and Mr. Smith and one desktop computer that belong to the secretary office.

The implementation of the information flow (which is depicted in figure 1.3) was based on the wireless networks of ICS-FORTH institute while the communication technologies that were used are SMS (short message service) messages via the GSM cellular network. More specifically, the individuals of the scenario are able to send requests, with their cell phones, to the secretary office

computer in the form of SMS messages and the office computer responds also via SMS messages. Moreover, the secretary office computer is equipped with the latest version of CDL application which is able to address authorization issues via DEAL language and some CDL algorithm extensions. Furthermore, in this scenario we make the same identification and authentication assumptions as in the previous one.

As for the policy of secretary office for answering a particular request, it is expressed below in CDL language. The rule *r1* indicates that a student *X* gets a degree, if he has passed all the degree lessons and has presented his thesis. Moreover, the rule *r1* implies that a classroom X is available at the time *Y*, if it does not exist any information of a presentation at the particular classroom and time. Finally, the rule *r3* indicates that the computer system has enough memory space, if the percentage of the used memory is smaller than 80%.

r_1^l: *getDegree(X)* ⇐ *passedLessons(X), presentedThesis(X)* .
r_2^l: *isAvaliable(X,Y)* ⇐ *not presentantionOn(X,Y).*
r_3^l: *enoughMemorySpace* ⇐ *usedMemoryBelow('80%').*

Moreover, the policy of the secretary office for authorizing a particular request is expressed below in DEAL language. The following rules of the authorization policy correspond to the authorization statements which are described in section 1.3.2.

<*deal1*> *belong(getDegree(X), 'StudentServices')* ← .
<*deal2*> *belong(getScholarship(X), 'StudentServices')* ← .

<*deal3*> *granted(X, 'StudentServices')* ⇐ *student(X).*
<*deal4*> ¬ *granted(X, 'StudentServices')* ⇐ *student(X),*
not registered(X).
<*deal5*> *superior(deal4, deal3)* ← .
<*deal6*> *granted(X, isAvaliable(Y,Z))* ⇐ *professor(X) ,*
not retired(X).
<*deal7*> *granted(X, enoughMemorySpace)* ⇐ *administrator (X).*

As for the knowledge base of the secretary office, it is provided below in DEAL language.

student('Bob') ← .

registered('Bob') ← .

getScholarship('Bob') ← .

student('Alice') ← .

passedLessons('Alice') ← .

presentedThesis('Alice') ← .

student('Trudy') ← .

professor('Antoniou') ← .

administrator('Smith') ← .

presentantionOn('RA201', 5) ← .

usedMemoryBelow('80%') ← .

Moreover, in this scenario there were preformed several requests. The replies to these requests verify that the system implements authorization control based on its authorization policy. To be more specific, we describe in detail the requests and the replies of the system.

First of all, we performed the query *getScholarship('Bob')* by Bob's cell phone to the secretary office. The secretary office authorization policy concluded that Bob is authorized for this request due to rules *deal2, deal3, deal4, deal5* and its knowledge base. Thus, the system proceeded on processing and answering the request. In conclusion, the secretary office replies to Bob the answer *'yes'* due to its local knowledge.

In addition, we performed the query *getDegree ('Alice')* by Alice's cell phone to the secretary office. The secretary office authorization policy concluded that Alice is authorized for this request due to rules *deal1, deal3, deal4, deal5* and its knowledge base. Thus, the system proceeded on processing and answering the request. In conclusion, the secretary office replies to Alice the answer *'yes'* due to CDL rule *r1* and its local knowledge.

Furthermore, we performed the query *getDegree ('Trudy')* by Trudy's cell phone to the secretary office. The secretary office authorization policy concluded that Trudy is not authorized for this request due to rules *deal1, deal3, deal4, deal5* and its knowledge base. Thus, the system did not proceed on processing and answering the request. Therefore, the secretary office replies to Trudy the answer *'undefined'*.

Additionally, we performed the query *isAvaliable('RA201', 5)* by professor's Antoniou cell phone to the secretary office. The secretary office authorization policy concluded that Antoniou is authorized for this request due to rule *deal6* and its knowledge base. Thus, the system proceeded on processing and answering the request. In conclusion, the secretary office replies to professor Antoniou the answer *'no'* due to CDL rule *r2* and its local knowledge.

Moreover, we performed the query *enoughMemorySpace* by Mr. Smith's cell phone to the secretary office. The secretary office authorization policy concluded that the administrator Smith is authorized for this request due to rule *deal7* and its knowledge base. Thus, the system proceeded on processing and answering the request. In conclusion, the secretary office replies to Smith the answer *'yes'* due to CDL rule *r3* and its local knowledge.

Overall, in the implementation of this scenario, the response of the secretary system is correct based on its authorization policy.

Conclusion

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

7.1 Synopsis

The special characteristics of ambient intelligence environments have introduced new research challenges in the field of authorization. The implementation of authorization policies is vital in order to develop a secure Ambient Intelligence system. Every AmI device should be able to specify access right policies to the resources that it controls. However, the imperfect nature of context information and the open and dynamic characteristics of AmI environments make the enforcement of authorization policies problematic.

The authorization problem has been addressed in many studies. Several authorization systems have been developed during the last 20 years both logic-based and non logic-based. However, most existing authorization systems are not appropriate to meet the demanding needs of Ambient Intelligence environments. More specifically, these approaches either don't support distributed authorizations or don't provide the expressive features of a powerful authorization language such as negative authorization, rule priorities, hierarchical category authorization and nonmonotonic reasoning.

This thesis studies, the problem of authorization in Ambient Intelligence environments. Firstly, it describes in detail the basic concepts of the authorization problem and the desirable characteristics of an authorization language for AmI environments. Secondly it proposes an approach that meets the predefined criteria. This approach proposes extensions to the framework presented in [32], which supports distributed contextual reasoning in ambient intelligence environments. The capabilities of this approach are illustrated using two fully implemented AmI scenarios that motivated our current research study.

In chapter 2 we provide the basic concepts of the authorization problem while in chapter 3 we describe the desirable characteristics of an authorization language for AmI environments. In chapter 6 we introduce the Distributed Environment

Authorization Language (DEAL) that aims on providing a powerful logic-based approach for addressing authorization issues in Ambient Intelligence environments. First of all, the chapter presents the syntax of DEAL by providing the alphabet, rules and characteristics of the language through examples. Secondly, it illustrates the semantics of the language through transformation to Defeasible Logic. Thirdly, it describes the appropriate extensions to the basic CDL algorithm in order to support DEAL policies. Finally, the chapter provides the implementation of the motivating scenarios using DEAL policies. Overall, this study suggests a formal high level logic-based authorization language which is sufficiently efficient to meet the increased authorization requirements in Ambient Intelligence environments.

7.2 Future Directions

This work is just one step in an ambitious research plan, and there are concrete ideas on further work for supporting the *DEAL* (*Distributed Environment Authorization Language*) approach.

First of all, future work on improving CDL implementation will also strengthen DEAL implementation because it is built as an extension to the current CDL application. The study in [37] provides several ideas for improving and extending CDL both in theoretical and in implementation level.

Secondly, our approach addresses the authorization problem and makes identification and authentication assumptions. However, the overall access control process requires strong identification and authentication techniques. Thus, further work can be conducted on the identification and authentication problem in order to be presented in conjunction with this work as a generic approach for access control.

Moreover, in this work distributed authorizations are supported by adopting the connection-based approach. Future work can be performed for also supporting distributed authorizations by adopting the credential-based approach. The advantages and disadvantages of both approaches are provided in chapter 4.

Finally, DEAL can be enriched with additional language characteristics that would empower its expressiveness. Some expressive features that could be added in DEAL are listed below. The addition of any of these features implies further research for the impact on the language complexity.

- *defeaters*: Defeaters are rules (which are supported in Defeasible logic and described in section 5.1) that they are used to block competing rules by deriving contrary evidence but they are not used independently to derive conclusions.
- *conflicting literals:* conflicting literals is a set of literals that any grounded pair of them forms a knowledge conflict. Obviously, given a predicate *p*, the set $\{p, \neg p\}$ is a set of conflicting literals. Other sets of conflicting literals may refer to discrete values that a variable can take, such as {on, off} and {hot, warm, cold}. A more detailed description of conflicting literals in an extended version of Defeasible logic is provided in [67].
- *conditional rule priorities:* conditional rule priority is a feature that enables the rule priorities to be expressed as a conclusion of a rule with a non-empty body. In DEAL language, conditional rule priorities could be supported if the priority rule could be specified as a defeasible or strict rule with non-empty conditions. The semantics of conditional rule priorities require changes in the defeasible reasoning process.

Overall, we believe that ambient intelligence environments provide a rich testbed for authorization approaches. Ambient Intelligence is a rich area with special requirements in terms of openness, distribution, heterogeneity and efficiency. Thus, it can serve as a source of inspiration for future work on the authorization problem.

Appendix A

A.1 TuProlog Reasoner

TuProlog [66] is a Java-based light-weight Prolog for Internet applications and infrastructures. For this purpose, tuProlog is designed to feature some interesting qualities: it is *easily deployable*, just requiring the presence of a Java VM and an invocation upon a single JAR file; its core is both *minimal*, taking the form of a tiny Java object containing only the most essential properties of a Prolog engine, and *configurable*, thanks to the loading and unloading of predicates, functors and operators embedded in libraries; the *integration between Prolog and Java* is as wide, deep, clean as possible; finally, *interoperability* is developed along the two main lines of Internet standard patterns and coordination models.

A.2 Defeasible Logic Metaprogram

On the TuProlog reasoning engine it is loaded the DR-PROLOG metaprogram. This metaprogram is actually a prolog program, implementing the defeasible logic, in a shorter more lightweight version of the original the ideas which are described in [48].

```
supportive_rule(Name, Head, Body) :- strict(Name, Head, Body).
supportive_rule(Name, Head, Body) :- defeasible(Name, Head,
Body).
rule(Name,Head,Body) :- supportive_rule(Name, Head, Body).


definitely(X):- fact(X).
definitely(X):- strict(R,X,L), definitely_provable(L).
definitely(X):- strict0(R,X,L), definitely_provable(L).
definitely(neg(X)):- strict1(R,neg(X),L),
definitely_provable(L), not(definitely(X)).
definitely(X):- strict2(R,X,L), definitely_provable(L).


definitely_provable([]).
```

```
definitely_provable(X):- definitely(X).

definitely_provable([X1|X2]):- definitely_provable(X1),
definitely_provable(X2).



defeasibly(X):- definitely(X).

defeasibly(X):- negation(X,X1), supportive_rule(R,X,L),
defeasibly_provable(L), not(definitely(X1)),
not(overruled(R,X)).

defeasibly_provable([]).

defeasibly_provable(X):- defeasibly(X).

defeasibly_provable([X1|X2]):- defeasibly_provable(X1),
defeasibly_provable(X2).



overruled(R,X):- negation(X,X1), supportive_rule(S,X1,U),
defeasibly_provable(U), not(defeated(S,X1)).

defeated(S,X):- sup(T,S), negation(X,X1),
supportive_rule(T,X1,V), defeasibly_provable(V).



negation(~(X),X):- !.

negation(X,~(X)).



append([],List,List).

append([Head|Tail],List2,[Head|Result]):-
append(Tail,List2,Result).

member(N,[N|Tail]).

member(N,[I|Tail]):- member(N,Tail).



minus_set([E|X],Y,Z):- member(E,Y),minus_set(X,Y,Z),!.

minus_set([E|X],Y,[E|Z]):-minus_set(X,Y,Z),not(member(E,Y)),
!.

minus_set([],Y,[]).

strict(e,w,r).

defeasible(y,t,e).

fact(w).
```

```
sup(e,w).
```

A.3 DEAL Metaprogram

On the TuProlog reasoning engine it is loaded the DEAL metaprogram. This metaprogram is actually a prolog program, implementing the DEAL language proof theoretic semantics.

```
%Nonmonotonic reasoning

defeasible(r1,  notExist(_H), [ ] ).

defeasible(r2,  ~(notExist(H)), [H] ).


%Transitivity of belong predicate

strict(r3,belongTo(X,Y),[ belong(X,Y) ]).

strict(r4,belongTo(X,Y),[ belong(X,Z), belongTo(Z,Y) ]).


%Derived authorizations among requester hierarchies

defeasible(r5, granted(X,Q) ,[ belongTo(X,Y), granted(Y,Q) ]).

defeasible(r6, ~(granted(X,Q)) ,[ belongTo(X,Y),
                                  ~(granted(Y,Q)) ]).

defeasible(r7, grant(G,X,Q) ,[ belongTo(X,Y), grant(G,Y,Q) ]).

defeasible(r8, ~(grant(G,X,Q)) ,[ belongTo(X,Y),
                                  ~(grant(G,Y,Q)) ]).


%Derived authorizations among query hierarchies

defeasible(r9, granted(X,Q) ,[ belongTo(Q,Y), granted(X,Y) ]).

defeasible(r10, ~(granted(X,Q)) ,[ belongTo(Q,Y),
~(granted(X,Y)) ]).

defeasible(r11, grant(G,X,Q) ,[ belongTo(Q,Y), grant(G,X,Y)
]).

defeasible(r12, ~(grant(G,X,Q)) ,[ belongTo(Q,Y),
~(grant(G,X,Y)) ]).
```

```
%Derived authorizations among action hierarchies

defeasible(r13, granted(X,right(A,O)) ,[ belongTo(A,Y),
granted(X,right(Y,O)) ]).

defeasible(r14, ~(granted(X,right(A,O))) ,[ belongTo(A,Y),
~(granted(X,right(Y,O))) ]).

defeasible(r15, grant(G,X,right(A,O)) ,[ belongTo(A,Y),
grant(G,X,right(Y,O)) ]).

defeasible(r16, ~(grant(G,X,right(A,O))) ,[ belongTo(A,Y),
~(grant(G,X,right(Y,O))) ]).


%Derived authorizations among object hierarchies

defeasible(r17, granted(X,right(A,O)) ,[ belongTo(O,Y),
granted(X,right(A,Y)) ]).

defeasible(r18, ~(granted(X,right(A,O))) ,[ belongTo(O,Y),
~(granted(X,right(A,Y))) ]).

defeasible(r19, grant(G,X,right(A,O)) ,[ belongTo(O,Y),
grant(G,X,right(A,Y)) ]).

defeasible(r20, ~(grant(G,X,right(A,O))) ,[ belongTo(O,Y),
~(grant(G,X,right(A,Y))) ]).
```

Bibliography

[1] P. Remagnino and G.L. Foresti. Ambient Intelligence: A New Multidisciplinary Paradigm. IEEE Transactions on Systems, Man and Cybernetics, Vol.35(1),pp 1-6, Jan. 2005.

[2] E. Aarts. Ambient intelligence: a multimedia perspective. Multimedia, IEEE , vol.11, no.1, pp. 12-19, Jan.-March 2004.

[3] C. Ramos, J.C. Augusto, and D. Shapiro. Ambient intelligence – the next step for artificial intelligence. IEEE Intelligent Systems, 23(2), 15–18, 2008.

[4] J.C. Augusto. Ambient Intelligence: Basic Concepts and Applications. Communications in Computer and Information Science, vol. 10, pp.14–24. Springer, Heidelberg (2008).

[5] G. Riva, F. Vatalaro, F. Davide and M. Alcañiz. Ambient Intelligence: From Vision to Reality. IOS Press, 2005.

[6] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a Better Understanding of Context and Context-Awareness. In HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, pages 304:307. Springer-Verlag, 1999. 1, 2

[7] B. Schilit and M. Theimer. Disseminating Active Map Information to Mobile Hosts. IEEE Network, 8(5):22{32, 1994. 1

[8] P. D. Gray and D. Salber. Modelling and Using Sensed Context Information in the Design of Interactive Applications. In EHCI '01: Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction, pages 317{336, London, UK, 2001. Springer-Verlag. 1

[9] N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced Reality Fieldwork: the Context-aware Archaeological Assistant. In V. Gaffney and M. van Leusen and S. Exxon, editor, Computer Applications in Archaeology 1997, British Archaeological Reports, Oxford, October 1998. Tempus Reparatum. 2

[10] K. Henricksen and J. Indulska. Modelling and Using Imperfect Context Information. In Proceedings of PERCOMW '04, pages 33{37, Washington, DC, USA, 2004. IEEE Computer Society. 2

[11] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In Proceedings of the 1996 IEEE Symposium on Security and Privacy, pages 164-173. IEEE Computer Society Press, May 1996.

[12] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance-checking in the PolicyMaker trust management system. In Proceedings of Second International Conference on Financial Cryptography (FC'98), volume 1465 of Lecture Notes in Computer Science, pages 254-274. Springer, 1998.

[13] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote Trust-Management System, Version 2, Internet Engineering Task Force RFC 2704, September 1999. http://www.ietf.org/rfc/rfc2704.txt

[14] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The role of trust management in distributed systems. In Secure Internet Programming, volume 1603 of Lecture Notes in Computer Science, pages 185-210. Springer, 1999.

[15] Y. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. REFEREE: Trust management for web applications. World Wide Web Journal, 2:706-734, 1997.

[16] D. Clarke, J. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certificate Chain Discovery in SPKI/SDSI, manuscript, Nov 1999.

[17] J. Elien. Certificate Discovery Using SPKI/SDSI 2.0 Certificates. Masters Thesis, MIT LCS, May 1998,
http://groups.csail.mit.edu/cis/theses/elien-masters.pdf

[18] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. Internet Engineering Task Force RFC 2693, September 1999. http://www.ietf.org/rfc/rfc2693.txt

[19] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. Simple Public Key Certificate, Internet Draft (Work in Progress), July1999. http://world.std.com/~cme/spki.txt

[20] R. L. Rivest, and B. Lampson. SDSI - A Simple Distributed Security Infrastructure, October 1996. http://theory.lcs.mit.edu/rivest/sdsi11.html

[21] S. Jajodia, P. Samarati, V. S. Subrahmanian, and Elisa Bertino. A unified framework for enforcing multiple access control policies. In Proceedings of ACM SIGMOD International Conference on Management of Data, pages 474-485, 1997

[22] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In Proceedings of the 1997 IEEE Symposium on Security and Privacy, pages 31-42. IEEE Computer Society Press, 1997.

[23] E. Bertino, F. Buccafurri, E. Ferrari, and P. Rullo. A Logical Framework for Reasoning on Data Access Control Policies. In Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW-12), pages 175-189, IEEE Computer Society Press, Los Alamitos, CA, 1999.

[24] S. Jajodia, P. Samarati, and V. S. Subrahmanian. Flexible Support for Multiple Access Control Policies. In ACM Transactions on Database Systems, Vol.26, No.2, June 2001, Pages 214-260

[25] N. Li, J. C. Mitchell, W. H. Winsborough. Design of a role-based trust management framework. In Proceedings of the 2002 IEEE Symposium on Security and Privacy, pp 114-130. IEEE Computer Society Press (2002).

[26] N. Li, B. N. Grosof, J. Feigenbaum. Delegation Logic: A logic-based approach to distributed authorization. ACM Transactions on Information and System Security (TISSEC), Vol 6(1): 128-171 (2003).

[27] N. Li, B. N. Grosof, J.Feigenbaum. A nonmonotonic delegation logic with prioritized conflict handling. Unpublished manuscript (2000).

[28] S. Wang, and Y. Zhang, A formalization of distributed authorization with delegation. In Proceedings of the 10th Australasian Conference on Information Security and Privacy (ACISP 2005), pp 303-315. Springer 2005.

[29] S. Wang and Y. Zhang, Handling distributed authorization with delegation through answer set programming. International Journal of Information Security. 6 (2007) 27-46.

[30] P. Liu, J. Hu, Z. Chen: A Formal Language for Access Control Policies in Distributed Environment. The 2005 IEEEWIC ACM International Conference on Web Intelligence WI05.

[31] A. Bikakis, G. Antoniou: Local and Distributed Defeasible Reasoning in Multi-Context Systems. In Proc. RuleML 2008: 135-149; an extended version of this paper has been conditionally accepted by Knowledge & Information Systems.

[32] A. Bikakis, G. Antoniou: Distributed Defeasible Contextual Reasoning in Ambient Computing. In Proc. AmI 2008: 308-325; an extended version of this paper has been accepted by IEEE Transactions on Systems, Man and Cybernetics.

[33] A. Bikakis, G. Antoniou: Contextual Argumentation in Ambient Intelligence. In Proc. LPNMR 2009: 30-43; an extended version of this paper has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering.

[34] G. Antoniou, D. Billington, G. Governatori, M.J. Maher: Representation results for defeasible logic. ACM Transactions on Computational Logic 2(2): 255-287 (2001).

[35] F. Giunchiglia, L. Serafini: Multilanguage hierarchical logics, or: how we can do without modal logics. Artificial Intelligence 65(1) (1994).

[36] G. Antoniou, C.Papatheodorou, A. Bikakis: On the Deployment of Contextual Reasoning in Ambient Intelligence Environments. In Proc. 6th International Conference on Intelligent Environments, 2010 (IE'10).

[37] A. Bikakis, G. Antoniou, P. Hassapis: Strategies for Contextual Reasoning with Conflicts in Ambient Intelligence, 2010, Knowledge and Information Systems (accepted) (2010)

[38] G.S. Graham and P.J. Denning. Protection | principles and practice. In Proceedings of the AFIPS Spring Joint Computer Conference, volume 40, pages 417{429, Atlantic City, New Jersey, May 16{18 1972.

[39] B.W. Lampson. Protection. In Proceedings of the 5th Princeton Symposium on Information Sciences and Systems, pages 437{443, Princeton University, March 1971. Reprinted in ACM Operating Systems Review, 8(1):18{24, January 1974.

[40] B.W. Lampson. A note on the con nement problem. Communications of the ACM, 16(10):613{615, October 1973.

[41] N. Li, W. H. Winsborough, J. C. Mitchell (2003) Distributed credential chain discovery in trust management. Journal of Computer Security, 11(1): 35-86.

[42] S. T. Kent. Internet privacy enhanced mail. Communications of the ACM, 36(8):48-60, August 1993.

[43] ITU-T Rec. X.509 (revised). The Directory - Authentication Framework. International Telecommunication Union, 1993.

[44] D. Clarke, J. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certificate chain discovery in SPKI/SDSI. Journal of Computer Security, 9(4):285-322, 2001.

[45] T.Y.C Woo, S.S. Lam, Authorizations in Distributed Systems: A New Approach. Journal of Computer Security, 2(2 & 3):107-136, 1993.

[46] E. Bertino, B. Catania, E. Ferrari and P. Perlasca, “A Logical Framework for Reasoning about Access Control Models”. ACM Transactions on Information and System Security, Vol.6, No.1, pp71-127, 2003.

[47] D. Nute. Defeasible logic. In D.M. Gabbay, C.J. Hogger, and J.A.Robinson, editors, Handbook of Logic in Artificial Intelligence and Logic Programming, Vol 3, pages 355-395. Oxford University Press, 1994.

[48] G. Antoniou, D. Billington, G. Governatori, M. J. Maher: Embedding defeasible logic into logic programming. TPLP 6(6): 703-735 (2006)

[49] J. McCarthy (1987) Generality in artificial intelligence. Commun ACM 30(12):1030-1035

[50] S. Buvac, IA Mason (1993) Propositional logic of context. In: AAAI, pp 412-419

[51] J. McCarthy, S. Buvač (1998) Formalizing context expanded notes. In: Aliseda A, van Glabbeek R, Westerståhl D (eds) Computing natural language. CSLI Publications, Stanford California pp 13-50

[52] C. Ghidini, F. Giunchiglia (2001) Local models semantics, or contextual reasoning = locality + compatibility. Artif Intell 127(2):221-259

[53] L. Serafini, P. Bouquet (2004) Comparing formal theories of context in AI. Artif Intell 155(1-2):41-67

[54] F. Roelofsen, L. Serafini (2005) Minimal and absent information in contexts. In: IJCAI, pp 558-563

[55] G. Brewka, F. Roelofsen, L. Serafini (2007) Contextual default reasoning. In: IJCAI, pp 268-273

[56] A. Casali, L. Godo, C. Sierra (2008) A logical framework to represent and reason about graded preferences and intentions. In: KR, pp 27-37

[57] J. Sabater, C. Sierra, S. Parsons, NR Jennings (2002) Engineering executable agents using multi-context systems. J Log Comput 12(3):413-442

[58] M. Dastani, G. Governatori, A. Rotolo, I. Song, L. van der Torre (2007) Contextual deliberation of cognitive agents in defeasible logic. In: AAMAS, p 148

[59] M. Cristani, E. Burato (2009) Approximate solutions of moral dilemmas in multiple agent system. Knowledge Information Systems 18(2):157-181

[60] G. Resconi, B. Kovalerchuk (2009) Agents' model of uncertainty. Knowledge Information Systems 18(2):213-229

[61] A. Bikakis, G. Antoniou: Rule-Based Contextual Reasoning in Ambient Intelligence. RuleML 2010, 74-88

[62] G. Governatori, M. J. Maher, D. Billington, and G. Antoniou. Argumentation Semantics for Defeasible Logics. Journal of Logic and Computation, 14(5):675-702, 2004. 29, 32, 36, 102, 105, 139

[63] M. J. Maher. A Model-Theoretic Semantics for Defeasible Logic. In Paraconsistent Computational Logic, pages 67{80, 2002. 139

[64] D. Billington, G. Antoniou, G. Governatori, M. J. Maher: Revising Nonmonotonic Theories: The Case of Defeasible Logic. KI 1999: 101-112

[65] M. J. Maher: Propositional Defeasible Logic has Linear Complexity. TPLP 1(6): 691-711 (2001)

[66] E. Denti, A. Omicini, A. Ricci, tuProlog: A Light-Weight Prolog for Internet Applications and Infrastructures, Practical Aspects of Declarative Languages, 3rd International Symposium (PADL 2001), Las Vegas, NV, USA, 11-12 March 2001. Proceedings. LNCS 1990, Springer-Verlag, 2001.

[67] D. Billington (1997) Conflicting Literals and Defeasible Logic, Proceedings of the Second Australian Workshop on Commonsense Reasoning in conjunction with the 10th Australian Joint Conference on Artificial Intelligence, 1997. 1-14.

[68] N. Li, B. N. Grosof, J. Feigenbaum: A Practically Implementable and Tractable Delegation Logic. IEEE Symposium on Security and Privacy 2000: 27-42

[69] N. Li, J. Feigenbaum, B. N. Grosof: A Logic-based Knowledge Representation for Authorization with Delegation. CSFW 1999: 162-174