DEAL: A Distributed Authorization Language for Ambient Intelligence

Irini Genitsaridi, Institute of Computer Science, F.O.R.T.H., Greece
Antonis Bikakis, Department of Information Studies, University College London
Grigoris Antoniou, Institute of Computer Science, F.O.R.T.H., Greece

ABSTRACT

Authorization is still an open problem in Ambient Intelligence environments. The difficulty of implementing authorization policies lies in the open and dynamic nature of such environments. The information is distributed among various heterogeneous devices that collect, process, change and share it. Previous work presented a fully distributed approach for reasoning with conflicts in ambient intelligence systems. This paper extends previous results to address authorization issues in distributed environments. First, we present the formal high-level authorization language DEAL to specify access control policies in open and dynamic distributed systems. DEAL has rich expressive power by supporting negative authorization, rule priorities, hierarchical category authorization and nonmonotonic reasoning. Then, we define the language semantics through Defeasible Logic. Finally, we demonstrate the capabilities of DEAL in a use case Ambient Intelligence scenario regarding a hospital facility, which served as our motivation.

Keywords: access control; distributed authorization; Ambient Intelligence; nonmonotonic reasoning; contextual reasoning; conflict resolution.

INTRODUCTION

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. AmI environments include heterogeneous intelligent devices that communicate by means of ad-hoc wireless networks. Each intelligent device acts as an autonomous entity that controls resources, handles requests and shares information and services with other entities. The core difference between AmI and traditional systems is the formers’ user centric approach. AmI systems adapt and respond to people by acknowledging their presence and gestures instead of the other way around.

Ambient Intelligence is a multidisciplinary approach as presented in [1, 31], since it requires the convergence of many areas of Computer Science in order to fulfill its purpose. Therefore, it has introduced new research challenges in many areas, including...
the field of access control. The implementation of access control is vital in order to develop a secure AmI system. Each intelligent device should be able to specify access policies to the resources that it controls. However, the special characteristics of AmI environments make the specification and implementation of access control problematic.

Ambient Intelligence environments are characterized by the imperfect nature of context information. Dey et al. [2] defined 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". Henrickssen and Indulska in [21] 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.

Moreover, AmI environments are characterized by their open and dynamic nature. In an open and dynamic environment participating entities enter or leave the environment at random times and without prior notice. Such entities are expected to have different goals, experiences and perceptive capabilities. They may use distinct vocabularies to describe their contexts, and may even have different levels of sociality. Due the unreliable and restricted (by the range of the transmitters) wireless communications, direct communication with all entities may not always be feasible.

In this paper we study the problem of authorization, as a basic part of access control in Ambient Intelligence environments, and provide a fully distributed approach to address it. Authorization is the process of specifying an access control policy that is used to determine whether a requester, with a given valid identity, is permitted to consume a particular requested service.

We propose a formal high-level logic-based language for addressing authorization issues in AmI environments. Our work builds on our previous work on a distributed model for contextual reasoning, called Contextual Defeasible Logic (CDL) [7,8,9]. CDL is based on Defeasible Logic [3,30], which is skeptical, rule-based, and uses priorities to resolve conflicts among rules. CDL also adopts ideas from Multi-Context Systems (MCS) [19]. A MCS consists of a set of logical theories called contexts, and a set of inference rules (a.k.a. mapping rules) that enable information flow between different contexts. In CDL, the Multi-Context Systems model is enriched through defeasible rules, and priority relations that provide a preference ordering on system contexts to represent their comparative reliability. Although CDL provides a flexible approach for reasoning about context in distributed environments, it does not address authorization issues. In this work we implement an authorization language as an extension of the language of CDL in order to address the access control requirements of Ambient Intelligence systems. We emphasize on the expressive power of the language in specifying authorization policies of distributed systems.

The rest of this paper is structured as follows. In the next section, we describe an Ambient Intelligence motivating scenario regarding a hospital facility emphasizing on its authorization requirements. Then we provide a thorough description of the authorization problem by presenting its basic concepts and the desirable characteristics of an authorization language. In the Background section, we describe the main features of Defeasible Logic and Contextual Defeasible Logic. Then we
introduce an authorization language, called DEAL; we present its syntax (alphabet and rules), main characteristics and semantics. In the last two sections we present a comparison with related approaches, summarize and discuss future research directions.

**MOTIVATING SCENARIO**

In this section we describe a motivating scenario from the ambient intelligence domain that refers to a hospital facility. We aim to automate hospital processes while preserving the safety of medical data.

**Description**

The hospital of the scenario consists of three autonomous departments: The Cardiology department for diagnostic heart or circulation services, the X-ray department for diagnostic imaging services, such as MRI (Magnetic resonance imaging), and the Gastroenterology department for diagnostic gastrointestinal services. Each department is equipped with a computer hosting a database server. The database stores information about equipment, medical examinations and personnel of the department.

The secretariat of the hospital hosts another database that stores administrative data about doctors, patients and more general information about diseases and the hospital departments.

All hospital computers are connected through a local wired network. Each computer is also equipped with Bluetooth, so that doctors can access their local data through their Bluetooth-enabled smartphones. Upon receiving a request for accessing their local data, department computers determine to accept / reject access based on the following access-control policy:

1. Doctors are authorized to access their patients’ examinations.
2. Trainees are authorized to access patient examinations if the patients’ doctors permit it.
3. Retired doctors are not authorized to access patients’ examinations.
4. Statement 3 is preferred to statement 1.

If the local information is not sufficient to determine whether the access request should be served, the computer collects relevant context information from other information sources, e.g. the secretariat database.

Consider the case that Alice, a pathologist working for the hospital, enters the cardiology department. Her smartphone establishes a connection with the Bluetooth computer and identifies the department. Then, as it has been set up by Alice, it reminds her about pending events that are relevant to the department, e.g. a reminder to ask for Bob’s (one of her patients) cardiology exams. Alice issues a request to access the exams through her smartphone. The department computer receives the request along with information about the device that issued the request (e.g. the name of its owner). It, then, issues a query to the secretariat computer asking for information about Alice’s status. When it gets the information that Alice is a doctor working for the hospital, it determines that access should be authorized to Alice and provides the results of Bob’s exams to her smartphone.

Charles is a trainee of Alice. In the same time that Alice is in the cardiology department, Charles visits the X-ray department. Following a similar procedure with Alice, Charles uses his smartphone to issue a request to access Bob’s X-rays exams. To determine about the acceptance of the request, the department computer sends a query to the secretariat computer asking for information about Charles. After receiving the information that Charles is a trainee of Alice, the department computer then attempts to contact Alice to get her permission to provide the results of Bob’s exams to Charles. Since Alice’s smartphone was located some minutes ago at the cardiology department, the permission request is delivered to Alice’s smartphone through the computer of the
department. Alice sends her permission through her smartphone and the local hospital network, and the X-ray department computer determines to provide Charles with access to Bob’s X-rays.

Dan is also a doctor who used to work for the same hospital but has recently retired. Before retiring, he was responsible for Alice’s patient, Bob. Dan can still use his smartphone to interact with the hospital computers. Through his smartphone he issues a request to the Gastroenterology department to access Bob’s gastroenterological examinations. The computer of the department gets information from the secretariat computer that Dan has just retired; therefore it rejects his request.

Figure 1: Context Information Flow

The context information flow of the scenario is depicted in Figure 1. Double arrows represent the users’ initial requests to access services, while single arrows represent requests to collect relevant information in the process of answering the service requests.

Assumptions and Challenges

The implementation of the scenario described above requires the combination of technologies from various fields including Human-Computer Interaction, Wireless Networks and Knowledge Representation and Reasoning (KRR). Our focus is on issues related to KRR. We implicitly make the follow simplifying assumptions for issues that are out of the scope of this paper.

- There is an available infrastructure for communication between the computers of the hospital and the doctors’ smartphones. The hospital computers communicate through the hospital local wired network, while communication between the computers and the doctors’ smartphones is enabled by Bluetooth.
- Each device is aware of parts of the knowledge that the other devices possess. For instance, each computer department is aware of the type of knowledge stored in the secretariat’s database.
- Each device has some minimum computing capabilities that are sufficient to conduct some simple reasoning tasks.
- Since our focus is on the authorization problem, we make identification and authentication assumptions. Specifically, we assume that the requesters are correctly identified and their identities are verified successfully when an intelligent device receives a request.

The specific problem that we focus on in this paper is:

“How to address distributed authorization in an environment characterized by distribution of the available relevant knowledge to several heterogeneous entities, taking into account the openness and dynamicity of the environment, the restrictions posed by wireless communications, and potential cases of missing or ambiguous information.”
THE AUTHORIZATION PROBLEM
In this section we describe the authorization problem in the domain of Ambient Intelligence. First, we analyze the basic concepts and notations of authorization. Then, we define the desirable characteristics of an authorization language for Ambient Intelligence environments.

Basic Concepts and Notations
In this subsection we define the concepts of request pair, authorization statement, authorization conflict and authorization policy.

Devices in Ambient Intelligence environments act as autonomous entities by sending and receiving requests from other entities. Using such requests, devices aim at consuming services that other entities provide. The pair that consists of the requesting entity and the requested service is called request pair.

In an Ambient Intelligence environment, authorization issues arise when an entity receives a request that is either sent from a device or is perceived through human interaction. An authorization statement (also known as authorization) expresses either permission - positive authorization - or denial - negative authorization - for a particular request pair.

The basic components of an authorization statement are listed below.

- **Service**: In an authorization statement the service has the form of a query (e.g. "open(File)?"). It is usually decomposed into two additional elements, an action and an object, that represent the right to perform an action on a resource object, and the resource itself, respectively.
- **Grantee**: The entity that receives the authorization for consuming a specific service. The grantee component may also refer to a group of entities indicating that every entity of the group receives the same authorization.

Another basic concept is the authorization conflict. An authorization conflict describes the problematic state where a positive and a negative authorization may be applied for the same grantor, grantee and service components. The resolution of an authorization conflict requires the specification of a preference among the contradictory authorizations. The following conflict resolution options could be used:

- **Denial-preference**: The negative authorization is preferred over the positive one. In this case, the grantor finally denies providing the service to the grantee.
- **Permission-preference**: The positive authorization is preferred over the negative one. In such case, the grantor permits the grantee to consume the service.
- **No-Preference**: Neither of the two authorizations is preferred thus the grantor neither permits nor denies the grantee to consume the service. The system may handle this case according to the specific application needs (e.g. as system error).

Given a particular request pair, the decision of whether the requester should be provided or denied the requested service is based on the authorization policy of the system. An authorization policy consists of a set of authorizations and conditions under which they are determined.

Logic-based approaches for the specification of authorization policies have been proven very successful since they offer significant advantages, e.g. simplicity, flexibility, formality, expressivity and modularity. In logic-based approaches, an authorization policy is defined as a set of logical rules. A logical rule that is contained in
an authorization policy is called authorization rule. An authorization rule can either be a final rule, which concludes to an authorization decision; or an intermediate rule, which specifies an intermediate conclusion. Two authorization rules are called conflicting (or contradictory) if their conditions can be simultaneously satisfied and their conclusions result in an authorization conflict.

**Desirable Characteristics**

In this subsection we describe in detail the desirable capabilities of an authorization language for distributed environments.

The expressive characteristics that should be provided by an authorization language are listed below.

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

A negative authorization expresses the denial of a grantor to provide a service to a grantee. The specification of negative authorizations is required in order to block specific request pairs. In the hospital scenario, negative authorization can be used to implement the third statement, which denies access to medical files to doctors who have retired. An authorization language should be able to support negative authorizations in order to easily specify such simple authorization statements.

Rule priorities is a feature that enables the specification of a priority relation over a set of rules. The priority relation can be used to denote a preference on a pair of conflicting rules. This feature is useful in many common scenarios that involve multiple authorization rules, which may potentially lead to inconsistencies (authorization conflicts). In the hospital scenario, statements 1 and 3 could lead to a contradiction (authorization conflict). Statement 4, however, can be encoded as a rule priority that gives preference to statement 3; therefore in case both statements (rules) may be applied for a given scenario, the system accepts only the conclusion of statement 3 as a valid conclusion. An authorization language should be able to support rule priorities in order to easily specify consistent policies with multiple authorization rules.

Hierarchical category authorization is a feature that enables the inheritance of authorizations, which are specified on hierarchical categories. Hierarchical categories express structured elements, which may refer to any relevant entities, such as users, devices, services, actions or objects. In our scenario, we can define the general class of doctors, as well as trainee doctors as a specific subclass of doctors. All authorizations that refer to a class of objects (e.g. permission to doctors for accessing medical data), are also applied to all its subclasses (e.g. permission to trainee doctors for accessing medical data). The components of an authorization that can be associated with hierarchical categories are: (a) the grantee; (b) the service; (c) the action; and (d) the object.

Nonmonotonic reasoning is a form of reasoning in which the acquisition of new knowledge can cause earlier conclusions to be withdrawn. It was developed to model commonsense reasoning used by humans. Such form of reasoning is supported by nonmonotonic logics, such as Defeasible Logic, where defeasible rules supporting contradictory conclusions may block each other’s conclusions, and rule priorities can be used to resolve such conflicts. Negation as failure is another type of nonmonotonic reasoning, where conclusions can be derived based on the absence of certain information. In our scenario, if there is no information about the retirement of Dan, the system will determine to provide him with the authorization to access Bob’s exams. When such information is entered into the system, such authorization will become invalid.
Distributed authorization is a form of authorization that may be based on both local knowledge and external information. Distributed authorizations are required in many cases in order to decrease authorization workload in an entity or in order to confirm an authorization decision by another trusted entity. In our motivating scenario, authorization to Charles is based on information residing in three different devices: (a) the X-rays department computer, which stores the local privacy policy; (b) the secretariat computer that stores information about doctors and their trainees; and (c) Alice’s smartphone, which provides the final permission. 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.

- **Connection-based** approach: This approach is based on runtime communications and information gathering from third-party entities. The authorizer must establish connections with the external entities that he wishes to communicate with, in order to receive relevant information.

- **Credential-based** approach: This 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 “Alice is a doctor” or more complicated policy statements. The credentials are usually provided to the authorizer by the requester, either together with the request, or at runtime according to the communication protocol.

The connection-based approach is a more 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 policy. Its main disadvantage 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. This approach is indirect since the authorizer receives the required information (in the form of credentials) usually from the requesting entity, which may not be related with the third-party entity that issued the credentials. It is also more static in the sense that it does not provide "fresh" information that is gathered during the process of the request, since credentials may have been issued any time earlier. The indirect and static nature of this approach results in more risks on the secure information flow.

Concluding this brief comparison, we argue that choosing the right approach depends on the specific requirements of the application. An interesting potential future direction would be to study a hybrid solution, which will combine the advantages of both approaches.

**BACKGROUND**

The authorization language that we propose in this paper has been developed on top of *Defeasible Logic* [3,30], and its distributed variant called *Contextual Defeasible Logic (CDL)* [7,8,9]. Before presenting the language in detail, below we give some background information on the two formalisms.

**Defeasible Logic**

Defeasible logic is a simple and efficient rule based non-monotonic formalism that was originally created by Donald Nute [30]. A thorough research on the formalism is also
provided in [3]. 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 a priority relation over the set of rules. In case of incomplete information, the logic is able to express defeasible conclusions; such conclusions can be withdrawn in the presence of new information.

A defeasible theory consists of three main elements: a set of facts, a set of rules and a superiority relation on the set of rules. Facts represent indisputable statements. Rules are classified into two categories: strict rules and defeasible rules (fuller versions of defeasible logics include also defeaters). Strict rules are “classical” rules in the sense that whenever their premises are indisputable (e.g., facts) then so are their conclusions. On the other, defeasible rules can be defeated (their conclusions are invalidated) by stronger contrary evidence. The superiority relation is a binary relation defined over the set of rules. The superiority relation determines which rule is stronger in case of a conflict between two competing rules.

Reasoning in Defeasible Logic is “skeptical”. This derives from the fact that when there is some support (a combination of facts and rules) for concluding $A$, but there is also support for concluding the negation of $A$ ($\neg A$), neither of the conclusions is derived, and the logic consults the priority relation to resolve the conflict. If the support for $A$ (or $\neg A$) has priority over the support for $\neg A$ (resp. $A$), then $A$ (resp. $\neg A$) is concluded.

Governatori et. al describe in [20] Defeasible Logic and its variants in argumentation theoretic terms. A model theoretic semantics is discussed in [29].

Contextual Defeasible Logic

Contextual Defeasible Logic implements a distributed nonmonotonic reasoning approach by combining ideas from Defeasible Logic and Multi-Context Systems. Multi-Context Systems [19] can be abstractedly defined as a set of contexts, which can be thought of as logic theories, and a set of inference rules (known as mapping or bridge rules) that enable information flow between different contexts.

In CDL, a MCS $C$ is defined as a collection of contexts. A context, $C_i$, is defined as a tuple $(V_i, R_i, T_i)$, where $V_i$ is the vocabulary used by $C_i$ (a set of positive and negative literals), $R_i$ is a set of rules, and $T_i$ is a partial preference ordering on $C$. $R_i$ consists of a set of local rules and a set of mapping rules. The body of a local rule is a conjunction of local literals (literals that are contained in $V_i$), while its head is labelled by a local literal. Local rules are classified into strict rules, and defeasible rules. Mapping rules associate local literals with literals from the vocabularies of other contexts (foreign literals). To deal with ambiguities caused by the interaction of mutually inconsistent contexts, mapping rules are also modelled as defeasible rules, while preference information from $T_i$ is used to resolve any potential conflicts.

DEAL LANGUAGE

In this section we present the formal high-level logic-based language DEAL (DistribEd Authorization Language) for expressing authorization policies in distributed environments. First, we describe its syntax. Then, we illustrate its expressive characteristics through examples from the hospital scenario. Finally, we provide the language semantics in detail.

DEAL Alphabet

The alphabet of DEAL consists of five sets of symbols: the constants ($C$), the variables ($V$),
the predicate symbols \((P)\), the logical symbols \((L)\) and the rule labels \((R)\).

Constants and variables are used in the classical sense. A constant has a specific non-changing value referring to an environment entity while a variable has a changing value that ranges over the set of constants \(C\). Constant symbols start with a lowercase letter while variable symbols start with an uppercase letter.

Predicate symbols are used to denote relations or properties of relevant entities. DEAL uses the following predicate symbols:

- **belongs\((X, Y)\)**: Represents that an element \(X\) belongs to a category of elements \(Y\). Moreover, it may represent that a category of elements \(X\) is a subcategory of category \(Y\). \(X\) and \(Y\) may take values either from \(C\) (sets of constants) or \(V\) (set of variables).
- **right\((X, Y)\)**: Represents the privilege to perform an action \(X\) to a resource \(Y\). Arguments \(X\), \(Y\) may also represent a category of actions and a category of resources, respectively. Both arguments may take values either from \(C\) or \(V\).
- **grant\((X, Y, Z)\)**: Represents a positive authorization (permission) that is given by a grantor \(X\) to a grantee \(Y\) for a service \(Z\). Arguments \(Y\), \(Z\) may also represent categories of entities and services respectively. All arguments may take values either from \(C\) or \(V\). Moreover, the service specified in \(Z\) may be represented by a right predicate.
- **granted\((Y, Z)\)**: Represents a positive authorization in the exact same sense as it is specified for predicate grant. The only difference is that the grantor is omitted as it is assumed to be the local system.
- **superior\((X, Y)\)**: Represents that given a pair of conflicting rules \((X, Y)\), the rule with label \(X\) is preferred to the rule with label \(Y\). Both \(X\) and \(Y\) take their values from the set of rule labels \(R\).
- **User defined predicates**: A user is able to define any predicate of \(n\)-arity in order to represent knowledge for a particular application domain. For example, if an application of 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\((X)\).

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

DEAL language supports the following logical symbols:

- **Strong Negation**: DEAL supports strong negation (a.k.a. classical negation) with the use of \(\lnot\) symbol. Strong negation can be used in front of any predicate to denote contradictory knowledge from what the predicate expresses. In other words, given a data element \(p\), which is a grounded predicate, then \(\lnot p\) represents the contradictory data element. Strong negation in front of grant or granted predicates expresses negative authorization.
- **Weak Negation**: DEAL supports weak negation (a.k.a. negation as failure) with the use of **not** keyword. Weak negation can be used in front of any predicate to denote the absence of the predicate from the knowledge base. 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, \(\lnot p\) may or may not be true. The difference between strong and weak negation for a predicate \(p\) is that the former \((\lnot p)\) represents the existence of negative (contradictory) information, while the latter \((\text{not} p)\) represents the absence of positive information about the predicate. The two definitions are not equivalent in an environment where \(p\) and \(\lnot p\) may coexist.
**Conjunction:** DEAL supports logical conjunction of literals with the use of comma (",") symbol.

**Strict entailment:** DEAL supports strict entailment with the use of the single line arrow, "⇒". Strict entailment can be used to express 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. Rules using strict entailment are called strict rules.

**Defeasible entailment:** DEAL supports defeasible entailment with the use of the double line arrow, "⇐". 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 side, whenever $X$ is true, $Y$ can be derived as a logical consequence, only if $¬Y$ cannot be derived by a preferred conflicting rule. Rules using defeasible entailment are called defeasible rules.

Rule labels is a set of symbols of the form $<\text{rule-x}>$, where rule-x takes its value from the set of constants. In DEAL, each rule is identified by its unique rule label.

**DEAL Rules**

In DEAL we distinguish four types of authorization rules: (a) final rules; (b) priority rules; (c) hierarchy rules; and (d) user-defined rules.

Direct knowledge (a.k.a. facts) is expressed by rules with empty body, while derived knowledge is expressed as conclusions of rules with non-empty body.

Below, we define the first three types of rules. User-defined rules are those that follow the syntax of DEAL rules but do not fall under any of the first three types of rules.

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

$$<\text{rule-label}> \text{ belongs}(X, Y) \leftarrow L_1, L_2, ..., L_n.$$  

The hierarchy rule supports the representation of hierarchical categories. The body of the rule is a conjunction of literals ($L_1$, $L_2$, ..., $L_n$) or the empty set. The rule concludes to a transitive relation. For example, consider the following hierarchy rules:

$$<\text{rule-1}> \text{ belongs}(a, b) \leftarrow .$$
$$<\text{rule-2}> \text{ belongs}(b, c) \leftarrow .$$

In this case, we conclude: $\text{belongs}(a, c)$.

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

$$<\text{rule-label}> G \leftarrow L_1, L_2, ..., L_n .$$

The final rule concludes to literal $G$ which represents a predicate from the set $\{\text{grant}, \text{granted}\}$ or their respective strong negations $\{\neg \text{granted}, \neg \text{grant}\}$, while $L_1$, $L_2$, ..., $L_n$ is a conjunction of any literals that are supported in DEAL or the empty set. In other words, the final rule concludes to an authorization, which is specified by $G$, while its fulfillment requirements are specified by the conjunction of literals $L_1$, $L_2$, ..., $L_n$. Moreover, the final rule supports hierarchical category authorization. For example, consider the following rules:

$$<\text{rule-1}> \text{ belongs}(a, b) \leftarrow .$$
$$<\text{rule-2}> \text{ granted}(b, q) \leftarrow .$$

In this case, we conclude: $\text{granted}(a, q)$.

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. Therefore, a final rule is specified as a defeasible rule, which can be blocked by a superior rule supporting a contradictory conclusion.
**Definition 3.** A priority rule is a rule of the following form:

\[
\text{<rule-label> superior(<r1>, <r2>)} \leftarrow .
\]

The priority rule is used to express a preference on a pair of conflicting rules (rules with contradictory conclusions). Given a pair of conflicting rules with labels \(<r1>, <r2>\), \(\text{superior(<r1>,<r2>)}\) denotes that rule labeled by \(<r1>\) is preferred to rule labeled by \(<r2>\). This actually means that in case that both rules can be applied, only the conclusion of rule \(<r1>\) can be derived. In case there is no priority relation associating the two rules, both their conclusions are blocked. In this way, inconsistency caused by contradictory conclusions is avoided.

The priority rule concludes to an acyclic relation. For example, the knowledge encoded by the following rules is considered invalid:

\[
\text{<rule-1> superior(<a>, <b>)} \leftarrow .
\]

\[
\text{<rule-2> superior(<b>, <c>)} \leftarrow .
\]

\[
\text{<rule-3> superior(<c>, <a>)} \leftarrow .
\]

**DEAL Characteristics**

In this subsection, we illustrate the expressive characteristics of DEAL through examples from the hospital scenario.

The authorization policy of the hospital departments is expressed by the following DEAL rules.

\[
\text{<deal1> granted(X, right(read, Z)) } \leftarrow \text{ belongs(X, doctors), belongs(Y, patients), treat(X,Y), examResults(Y, Z)} .
\]

\[
\text{<deal2> granted(X, right(read, Z)) } \leftarrow \text{ belongs(X, trainees), belongs(Y, patients), belongs(W, doctors), treat(W,Y), examResults(Y, Z), grant(W, X, right(read, Z))} .
\]

\[
\text{<deal3> } \neg \text{ granted(X, right(read, Z)) } \leftarrow \text{ belongs(X, retDoctors), belongs(Y, patients), examResults(Y, Z)} .
\]

\[
\text{<deal4> belongs(retDoctors, doctors) } \leftarrow .
\]

\[
\text{<deal5> superior(deal3, deal2) } \leftarrow .
\]

Rules \text{deal1} and \text{deal2} correspond to authorization statements 1 and 2 of the motivating scenario, respectively. Rules \text{deal3} and \text{deal4} correspond to statement 3, while rule \text{deal5} corresponds to statement 4.

Rules \text{deal1} and \text{deal2} express positive authorization (permission) for doctors and trainees to access patients’ examinations, while \text{deal3} expresses negative authorization (denial) for retired doctors. Rule \text{deal5} expresses a rule priority of \text{deal3} over \text{deal2}. Finally, \text{deal4} specifies the class of retired doctors as a subclass of doctors (hierarchy categorization).

The same policy could by implemented by replacing rules \text{deal1}, \text{deal3} and \text{deal5} with rules \text{deal6} and \text{deal7}.

\[
\text{<deal6> granted(X, right(read, Z)) } \leftarrow \text{ belongs(X, doctors), belongs(Y, patients), treat(X,Y), examResults(Y, Z), not belongs(X, retDoctors)} .
\]

\[
\text{<deal7> } \neg \text{ granted(retDoctors,right(read, Z)) } \leftarrow \text{ belongs(Y, patients), examResults(Y, Z)} .
\]

Rule \text{deal6} uses negation as failure in its body, while \text{deal7} expresses hierarchical category authorization through the derivation of the negative authorization (denial) to all elements of the category \text{retDoctors}. Note that the two rules cannot lead to authorization conflicts because their premises cannot be simultaneously satisfied for the same grantee.

Finally, rules \text{deal1}, \text{deal2}, \text{deal3}, \text{deal6} and \text{deal7} implement distributed authorization in the sense that their evaluation requires collecting information from one or more
remote devices. For instance, deal2 requires the evaluation of (a) belongs and treat
predicates by the secretariat department; (b) examResults predicate by the Xray
department; and (c) a grant predicate by Alice’s PDA.

**DEAL Semantics**

In this subsection we illustrate the semantics of DEAL through transformation into
Defeasible Logic (DL) and CDL.

DL supports constants and variables exactly as they are specified in DEAL. It also
supports rules identification using unique rule labels. DL also enables the specification of
user-defined predicates. Therefore, all DEAL predicates can be implemented in DL.
Moreover, DL supports directly the features of strong negation, logical conjunction,
and strict and defeasible entailment, as they are defined in DEAL. Weak negation is not directly
supported by DL, but can be simulated using a technique based on auxiliary predicates, which
was first presented in [4]. Specifically, every declaration of not X, where X is a language
literal, can be equivalently replaced with not(X) (where not is an auxiliary predicate) and the addition of the following two rules.

\[ r1: \quad \text{not}(X) \iff \]. \]

\[ r2: \quad \neg \text{not}(X) \iff X. \]

Furthermore, all DEAL rules can be translated into DL. The priority rules of
DEAL are implemented in DL using the acyclic superiority relation on the set of rules.
A superiority relation is defined with the use of “>”. Given a rule name r1 at the left side of
the operator and a rule name r2 at the right side, it is denoted that r1 is preferred to r2. If
in a defeasible theory D both rules may be applied, only the conclusion of r1 can be
derived from D.

The hierarchy rule is specified in DL exactly as in DEAL, while the transitivity of
the belongs relation is supported with the addition of the following two rules.

\[ \text{belongs}(X, Y) \iff \text{belongsTo}(X, Y). \]

\[ \text{belongs}(X, Y) \iff \text{belongsTo}(X,Z), \]

\[ \text{belongs}(Z, Y). \]

The auxiliary belongsTo predicate is used to meet the requirement that the relation
expressed by belongs should be acyclic, so as to avoid loops in DEAL policies.

Final rules are specified in DL exactly as in DEAL, while hierarchical category
authorization is implemented using the following four sets of DL rules. The first set
implements grantee hierarchies.

\[ \text{granted}(X, Q) \iff \text{belongs}(X, Y), \]

\[ \text{granted}(Y, Q). \]

\[ \neg \text{granted}(X, Q) \iff \text{belongs}(X, Y), \]

\[ \neg \text{granted}(Y, Q). \]

\[ \text{grant}(G, X, Q) \iff \text{belongs}(X, Y), \]

\[ \text{grant}(G, Y, Q). \]

\[ \neg \text{grant}(G, X, Q) \iff \text{belongs}(X, Y), \]

\[ \neg \text{grant}(G, Y, Q). \]

The second set service hierarchies.

\[ \text{granted}(X, Q) \iff \text{belongs}(Q, Y), \]

\[ \text{granted}(X, Y). \]

\[ \neg \text{granted}(X, Q) \iff \text{belongs}(Q, Y), \]

\[ \neg \text{granted}(X, Y). \]

\[ \text{grant}(G, X, Q) \iff \text{belongs}(Q, Y), \]

\[ \text{grant}(G, X, Y). \]

\[ \neg \text{grant}(G, X, Q) \iff \text{belongs}(Q, Y), \]

\[ \neg \text{grant}(G, X, Y). \]

The third set implements action hierarchies.

\[ \text{granted}(X, \text{right}(A,O)) \iff \text{belongs}(A,Y), \]

\[ \text{granted}(X, \text{right}(Y,O)). \]

\[ \neg \text{granted}(X, \text{right}(A,O)) \iff \text{belongs}(A, Y), \]

\[ \neg \text{granted}(X, \text{right}(Y,O)). \]

\[ \text{grant}(G, X, \text{right}(A,O)) \iff \text{belongs}(A, Y), \]

\[ \text{grant}(G, X, \text{right}(Y,O)). \]

\[ \neg \text{grant}(G, X, \text{right}(A,O)) \iff \text{belongs}(A, Y), \]

\[ \neg \text{grant}(G, X, \text{right}(Y,O)). \]
The last set implements object hierarchies.

\[
\text{granted}(X, \text{right}(A,O)) \iff \text{belongs}(O, Y), \\
\text{granted}(X, \text{right}(A,Y)).
\]

\[
\neg \text{granted}(X, \text{right}(A,O)) \iff \text{belongs}(O, Y), \\
\neg \text{granted}(X, \text{right}(A,Y)).
\]

\[
\text{grant}(G, X, \text{right}(A,O)) \iff \text{belongs}(O, Y), \\
\text{grant}(G, X, \text{right}(A,Y)).
\]

\[
\neg \text{grant}(G, X, \text{right}(A,O)) \iff \text{belongs}(O, Y), \\
\neg \text{grant}(G, X, \text{right}(A,Y)).
\]

Finally, distributed authorization is implemented using the notion of contexts and mapping rules of CDL. Specifically, each different entity (e.g. each computer / device in the hospital scenario) is defined as a context in a Multi-Context System \( C \). Each rule that combines both local knowledge of the entity (local context) and foreign knowledge from other contexts in \( C \) is defined as a mapping rule of the local context. In the hospital scenario, rules deal1, deal2, deal3, deal6 and deal7 are implemented as mapping rules of the cardiology, X-rays and gastroenterology departments. For example, deal1, which is used by the cardiology department to determine access rights for doctors, is defined as a mapping rule of the form:

\[
<\text{deal2}> \quad \text{granted}(X, \text{right}(\text{read}, Z))_C \iff \\
\text{belongs}(X, \text{doctors})_S, \\
\text{belongs}(Y, \text{patients})_S, \\
\text{treat}(W,Y)_S, \\
\text{examResults}(Y, Z)_C
\]

The above rule combines knowledge from the local context \((\text{examResults}(Y, Z)_C)\) with foreign knowledge from other system contexts, namely the secretariat \((\text{belongs}(X, \text{doctors})_S, \text{belongs}(Y, \text{patients})_S, \text{treat}(W,Y)_S)\), to conclude about a local context conclusion \((\text{granted}(X, \text{right}(\text{read}, Z))_C)\). The subscript at the end of each literal name is used to denote the context that the respective literal is defined by.

**RELATED WORK**

Over the past twenty years, several authorization approaches have been proposed for distributed environments. In this chapter we present the approaches that are related to our work and describe their main limitations.

The trust-management approach, which was initially proposed by Blaze et al. in [13], is focused on the credential-based method for distributed authorization and views the authorization decision as a “proof-of-compliance” problem: *Does a set of credentials prove that a request complies with a policy?* The frameworks of PolicyMaker [13, 12], REFEREE [14], Keynote [10, 11] and SPKI/SDSI [15, 16, 17, 18] are more recent attempts towards a trust management framework. As they are not based on formal logics, most of them do not provide declarative semantics and lack important expressive characteristics. The first three approaches do not support negative authorization, whereas the last one does not support conjunction of attributes and attributes with fields.

On the other hand, logic-based authorization methodology is a very flexible and declarative approach that achieves separation of authorization policies from implementation mechanisms and provides policies with precise semantics. The logic-based authorization approaches that are proposed by Jajodia et al. in [22, 23, 24] and Bertino et al. in [5, 6] are quite expressive. Jajodia et al. proposed the Flexible Authorization Framework (FAF) that incorporates an authorization specification logic language (ASL). ASL can be used to encode the system security needs. ASL supports negative authorizations, hierarchical category authorizations, and nonmonotonic reasoning through the use of negation as failure. Bertino et al. proposed a logic formalism for expressing authorization policies, enabling features such as hierarchical category authorization, negative
authorizations (through the use of strong negation) and nonmonotonic reasoning (through the use of negation as failure). The main limitation of both approaches is that they do not support distributed authorization; they are more focused on centralized applications.

Decentralized logic-based approaches have also been proposed in the recent literature [25,26,28,32,33]. Compared to DEAL their main limitations are: (a) [26,27] do not support negative authorization, nonmonotonic reasoning and rule priorities; (b) The D2LP language proposed in [25], which implements the nonmonotonic version of [26], and the nonmonotonic framework FACL4DE proposed in [28], do not support hierarchical category authorization; (c) the AL language proposed in [32,33] does not integrate any type of priority / preference information.

In conclusion, logic-based approaches have been proven very successful in specifying authorizations. However, none of the existing authorization logic-based approaches combines all desirable characteristics dictated by the real needs of Ambient Intelligence environments.

CONCLUSIONS AND FUTURE WORK
To conclude this paper, we summarize and discuss its main contributions, and propose possible directions for future research.

This paper studies the problem of authorization in Ambient intelligence environments. First, it describes in detail the basic concepts of the authorization problem and the desirable characteristics of an authorization language for AmI environments. Then, it proposes an approach that meets the predefined criteria. It introduces the formal high level logic-based language, DEAL for addressing authorization issues in Ambient Intelligence environments. The syntax and semantics of DEAL language are described thoroughly. Moreover, it provides an authorization scenario from the Ambient Intelligence domain that is used to demonstrate the expressive power of the language.

This work is just one step in an ambitious research plan, and there are concrete ideas on further work. Our approach addresses the authorization problem by making identification and authentication assumptions. However, the overall access control process requires strong identification and authentication techniques. It is among our priorities to combine our framework with appropriate methods and techniques from this field. Another interesting future direction would be to study a hybrid solution of the connection-based and credential-based approach for supporting distributed authorization. Finally, DEAL can be enriched with additional language characteristics that would empower its expressiveness such as defeaters and conflicting literals, which are supported by alternative versions of Defeasible Logic.

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. Therefore, it can serve as a source of inspiration for future work on the authorization problem.

REFERENCES


