Chapter 9
Argumentative Agents for Service-Oriented Computing

M. Morge, J. McGinnis, S. Bromuri, P. Mancarella, K. Stathis, and F. Toni

Abstract. We propose an argumentation-based agent model that supports service and partner selection in service-oriented computing settings. In this model, argumentation is also used to help agents resolve conflicts between themselves, whenever negotiation is required for the provision of complex services. The model relies upon an argumentation framework that is used in a modular architecture where Knowledge, Goals, Decisions and Priorities are manipulated by three specialized modules dealing with decision making, communication and negotiation. We formulate a distributed e-procurement process to illustrate how our agents select services and partners and can negotiate with one another.

Maxime Morge
Université Lille 1, France
e-mail: Maxime.Morge@univ-lille1.fr

Jarred McGinnis
Press Association, London UK
e-mail: Jarred.Mcginnis@pressassociation.com

Stefano Bromuri
University of Applied Science, Western Switzerland
e-mail: Stefano.Bromuri@hevs.ch

Paolo Mancarella
Università di Pisa, Italy
e-mail: Paolo.Mancarella@unipi.it

Kostas Stathis
Royal Holloway, University of London, UK
e-mail: Kostas.Stathis@cs.rhul.ac.uk

Francesca Toni
Imperial College London, UK
e-mail: ft@doc.ic.ac.uk

M. Ganzha & L. C. Jain (Eds.): Multiagent Systems & Applications, ISRL 45, pp. 217–255.
springerlink.com

1 Introduction

Service-oriented computing (SOC) is an emerging inter-disciplinary paradigm for distributed computing, which is changing the way software applications are developed, deployed and utilised. The central theme of service-oriented computing are services that provide autonomous, platform-independent, computational elements that can be described, published, discovered, orchestrated and programmed using standard protocols to build networks of collaborating applications distributed within and across organizational boundaries.

The underlying principles and methodologies of SOC assume a service-oriented architecture (SOA). A basic SOA is often understood as defining an interaction between software agents as an exchange of messages between service requesters (clients) and service providers [50]. Clients are software agents that request the execution of a service. Providers are software agents that provide the service. Agents can be simultaneously both service clients and providers. Providers are responsible for publishing a description of the service(s) they provide. Clients must be able to find the description(s) of the services they require and must be able to bind to them. The basic SOA is not an architecture only about services, it is a relationship of three kinds of participants: the service provider, the service discovery agency, and the service requestor (client).

One of the main issues for SOC applications is how to compose services by assuming an open environment where the relationship of service providers, clients and discovery agencies change over time as new agents enter and/or leave the application dynamically. The issue here is how to establish a possible service composition by negotiating terms and conditions of the participating services for a given duration and according to the requirements and the goals the clients and the providers have to satisfy. The problem here is how to develop mechanisms that allow this negotiation to take place by reducing possible conflicts to support successful service compositions.

To address the issues involved in negotiating complex and composite services, we present an argumentation-based model of agency and its associated architecture to support SOC applications. Argumentation is used to support the agent to reason about services, engage in communicative interaction with other agents to negotiate desired service properties, resolve conflicts and eventually make decisions of which service or partner to select. Argumentation's main strength is that it allows an agent to plead for and against conclusions [52], providing a powerful deliberative and dialectical model for interacting, decision-making agents to assess the validity of received information and to be able to resolve conflicts and differences of opinion. It is an essential ingredient of decision-making [29, 4, 7, 45, 48], inter-agent communication [40], and negotiation [53, 30, 2].

The proposed agent model was developed within ArguGRID, an EU-funded project[1] that started in 2006 and ended in 2009. ArguGRID has also developed a multi-agent systems platform, a peer-to-peer infrastructure and an environment for users to interact with agents and the overall system, by authoring workflows of

[1] http://www.argugrid.eu/

services, hosted upon a Grid platform. While presenting the agent model, we will also emphasise the links between this and the other components of the ArguGRID system.

As shown in Figure 1, argumentation agents in ArguGRID are deployed within a Grid/Service-oriented platform to represent service clients and providers. The platfom consists of four main interacting components.

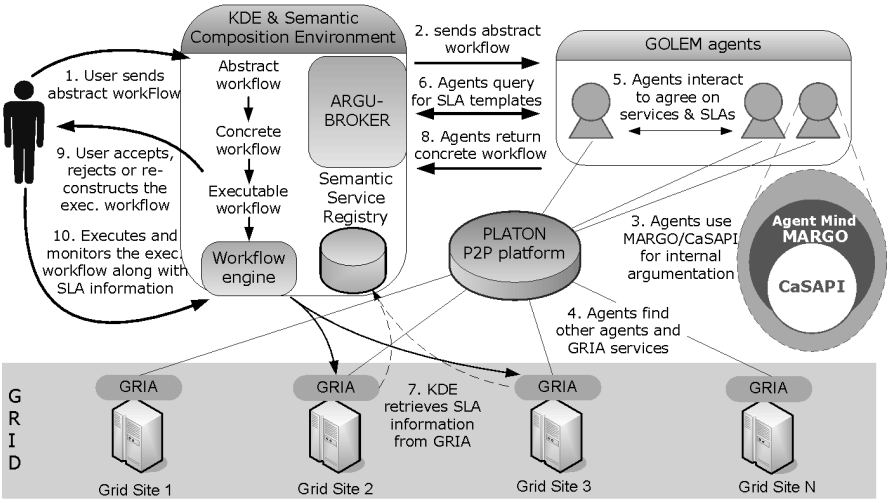

Fig. 1 ArguGRID Platform

KDE is a commercial software tool developed by InforSense Ltd[2]. This system provides facilities to build end user application as remote Web services or Grid services [24]. For the needs of ArguGRID, the KDE system has been extended to support semantic descriptions. In this way, requester goals representing user requirements can be matched with concrete services and can be executed within the Grid infrastructure. KDE was chosen as a supporting technology for the ArguGRID project because InforSense was the main developer of KDE and ArguGRID had several exploitation advantages from the perspective of a company working on workflow management systems.

GRIA[3] is the GRID middleware that ArguGRID has chosen to use to support its scenarios. GRIA is a service-oriented infrastructure designed particularly to support Business-to-Business collaborations (such as the ones required by the ArguGRID scenarios) through service provision across organisational boundaries in a secure, interoperable and flexible manner. In particur GRIA was chosen as it is one of the main technologies to represent Web services on the Grid.

[2] http://www.inforsense.com

[3] http://www.gria.org

PLATON[4] (Peer-to-Peer Load Adjusting Tree Overlay Networks) is a Peer-to-Peer platform supporting multi-attribute and range queries [36] which has been developed in the ArguGRID project to support service discovery mechanisms for load-balancing of peer resources. Load-balancing of peer resources is necessary in order to guarantee logarithmic querying time using any distributed tree-based multi-attribute Peer-to-Peer platform. The reason to include a P2P platform in the project was to allow ArguGRID to scale up to possibly thousands of services. In particular PLATON allows to perform multidimensional queries on our Web services, meaning that we can represent Web services in terms of complex descriptions as we will see later in Section 6.

GOLEM[5] (Generalized OntoLogical Environments for Multi-agent systems) is an agent environment middleware [10] which has been developed in the ArguGRID project. In this chapter, we present how to use GOLEM to host MARGO agents. The deployment of these agents will be described in Section 6. The GOLEM agent platform was chosen as it allows a simple integration between agents and external entities represented as objects in the GOLEM agent environment. In particular, GOLEM has been helpful to represent Web services in terms of objects accessible by the agents of ArguGRID.

As shown in Figure 1, the operation of the platform from the point of view of service requestors can briefly be explained as follows:

1. a user develops an abstract workflow reflecting his/her requirements using the KDE;
2. the abstract workflow is then communicated to the agent that reasons about services and makes decisions, aiding the refinement process of the abstract workflow;
3. the reasoning capabilities of the agent are based on using the MARGO argumentation engine for decision-making, which in turn uses the CaSAPI general-purpose argumentation engine;
4. in order to discover an appropriate agent or a GRIA Grid service, agents are given the capability to use the P2P platform, linking all available agents and GRIA services in a virtual registry that can be queried;
5. an agent can negotiate with other agents to find a concrete workflow whose execution will satisfy the application requirements, as stated in the abstract workflow;
6. agents can query the KDE system about SLA (Service Level Agreement) templates, needed for creating a concrete workflow that satifies user preferences.
7. the KDE, by accesing GRIA nodes, can retrieve information about SLA's;
8. having carried out its mission, the agent representing the user (i.e. the initial agent that received the abstract workflow from the KDE) will return to the KDE the concrete workflow, constituted of a set of GRIA services to be executed in a certain manner/sequence;

[4] http://platonp2p.sourceforge.net
[5] http://www.golem.cs.rhul.ac.uk

9. the KDE shows the suggested workflow to the user. If the user rejects the suggestion, a new workflow would be created;
10. the KDE executes and monitors the resulting workflow. Details about execution and the final results are presented to the user.

Given the ArguGRID context, the main focus and contribution of this work is an argumentation-based agent mind applied to service selection and partner selection in an e-procurement scenario as described in [58]. The e-procurement scenario illustrates how agents support the selection and provision of services for their integration in an open and distributed environment. In this context, a human user requesting a service is only required to specify an abstract description of his needs for these services, possibly with some constraints and preferences about them. The selection of these services, as well as the selection of partners, are tasks delegated to the autonomous agents.

The mind of our agents is composed of three main modules: (i) the *individual decision making* module, allowing to translate the goals, preferences, and constraints provided by the user requesting a service to an internal and abstract representation of the user's needs (ii) the *social decision making* module, allowing to move, by means of negotiation, from these abstract representations to concrete ones, in terms of contracts; (iii) the *social interaction* module, managing the communication given social rules of interaction.

Note that the services available in ArguGRID are encapsulated in Web services and available to the agents deployed in the ArguGRID platform. In this chapter we do not focus on how these services are implemented, but we rather focus on how the agents can select these services, according to their characteristics, by means of argumentation. Thus, we are not concerned with defining semantic Web reasoners or ontologies to describe Web services. We have assumed a fixed ontology that is common to the agents involved in the procurement process and in order to focus on argumentation, communication protocols and the reasoning process.

The remainder of this chapter is organised as follows: Section 2 provides a detailed discussion about the motivating e-procurement scenario; Section 3 discusses the background formalisms and technologies on which we built the agent interaction in our ArguGRID e-procurement scenario; Section 4 provides the architecture of our cognitive model; Section 5 illustrates the agents' deliberation and communication capabilities with a case run of a e-procurement scenario; Section 6 describes implementation issues related to the deployment of our multi-agent system; Section 7 discusses related work; finally, Section 8 concludes by summarising our proposal and discussing our future plans.

2 Motivation: An E-Procurement Scenario

In order to illustrate our model and architecture, we consider an e-procurement scenario, introduced in [58], where a buyer seeks to purchase combined services/products from A-type and B-type suppliers. The e-procurement scenario is particularly suitable for our evaluation as it implies the use of distributed agents representing

distributed services that require to reach an agreement by means of negotiation protocols between buyers and suppliers. The suppliers combine their competencies in order to provide solutions for the buyer. Each agent is representing a user, i.e. a service requester or a service provider. Then, each agent may be responsible for many different Web services. This operation is typically achieved by means of a 6-steps procurement process whereby: a requester looks for potential suppliers (step 1), gathers information in order to evaluate the potential suppliers (step 2), creates a short-list according to this information (step 3). The requester asks the short-listed suppliers to provide a quote for the services (step 4), chooses one of the suppliers (step 5), and finally the requester and the winner negotiate the terms & conditions of the contract, such as the price and the warranty (step 6). If this last step is unsuccessful, the process goes back to the step 5. When no suitable service is found, the requester agent asks its user to reconsider his needs by relaxing the constraints. Figure 2 summarises these steps and indicates the types of dialogues required to support them. These dialogues should conform to suitable protocols.

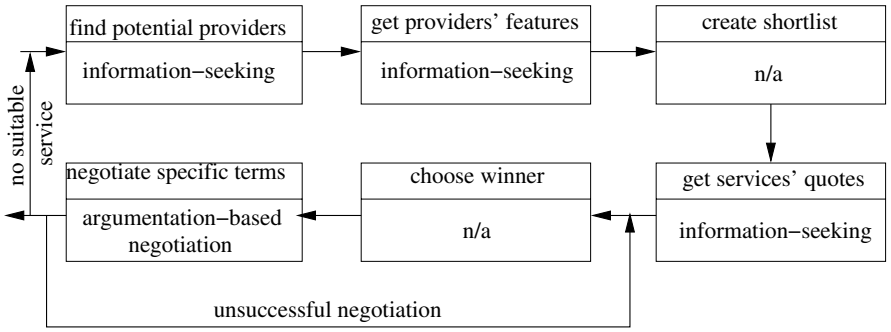

Fig. 2 Deliberative steps for e-procurement

The exchange of offers and acceptances may lead to contracts, i.e. legal relations between providers and requesters that typically force commitments (e.g. obligations) from one agent to another about the provision of services. In this chapter, we will assume that contracts are simple transactions between a provider and a requester, characterised by features of the services provided, and will ignore complex issues such as enforcement of commitments, sanctions, penalties, etc. Our definition of contract is given in Section 4.3. The contracts create virtual organisations (VO, for short) consisting of the buyer, the selected A-type agent and the selected B-type agent.

Within the e-procurement scenario we consider a specific case where a buyer looks for an e-ordering system (S) which is composed of a computer system (S_a) and an Internet connection (S_b). The A-type agents providing S_a are `Al` and `Alice`. While `Alice` is responsible for the concrete instances of services $S_a(a_1)$ and $S_a(a_2)$, `Al` can provide $S_a(a_3)$ and $S_a(a_4)$. The B-type agents providing S_b are `Bob`

and Barbara. Bob is representing the concrete services $S_b(c)$, $S_b(d)$, $S_b(e)$ and $S_b(f)$. On the other hand, Barbara is representing the services $S_b(g)$ and $S_b(h)$. A-type agents are responsible to select B-type agents and provide the combined S to the buyer (cf Figure 3). A possible VO in this case may consist of Bob, Al and the buyer, with two contracts: the first is between the buyer and Al and concerns the combined service S, the second is between Al and Bob and concerns service S_b. These contracts may be negotiated as follows. At first, the buyer plays the role of requester and all A-type agents play the role of potential suppliers in an e-procurement process for service S. In turn, the A-type agents play the role of requesters and the B-type agents play the role of potential suppliers in an e-procurement process for service S_b.

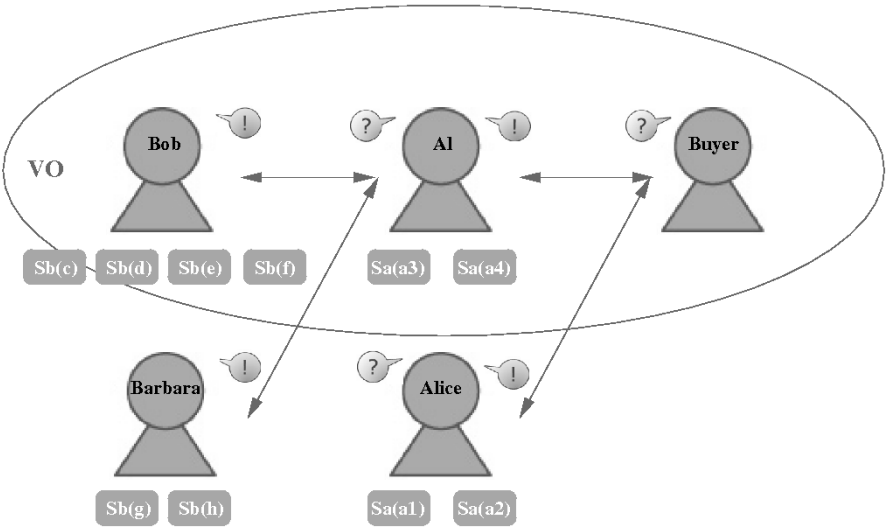

Fig. 3 Use case scenario

In the remainder of this chapter, we will focus on the e-procurement process involving Al as a requester, and Bob as a supplier. Al's goal consists of finding and agreeing to a service S_b provided by a B-type agent. According to its preferences and constraints:

- the cost of the service must be low (e.g. within a budget of 10 euros per month), and
- the quality of the service must be high (e.g. with warranty of a 24 hours assistance).

Taking into account its goal and preferences/constraints, Al needs to solve a decision-making problem where the decisions amount to a service and a supplier for that service. On the other hand, the goal of the supplier Bob consists of a decision-making problem about providing a service. According to Bob's preferences and constraints:

- the cost of the service must be high (e.g. within a budget of 40 euros per month), and
- the quality of the service must be low (e.g. with warranty of a 2 hours daily assistance).

A negotiation is a multi-step interaction between Al and Bob by means of which the agents aim at achieving the common goal to make a deal while resolving their conflicting interests about the cost and the quality of the service. The main strength of the argumentation-based approach we will use is to provide extra information during the interaction. For this purpose, we will assume the agents share a set of common ontologies to describe the services (in particular the intervals in which the price and warranty can be contracted). Moreover, argumentation-based decision making allows agents to identify and resolve the possible conflicts. The two decision making processes of the participants take place in a dynamic setting, whereby information about other agents, the services they require/provide and the characteristics of these services are obtained incrementally within the e-procurement process outlined in Figure 2. As we will see later on, the outcome of this process is the contract obliging Bob to provide a service S_b to Al, with low cost (e.g. 10 euros per month) and low quality (e.g. a warranty of a 2 hours daily assistance).

Within our proposed agent model and architecture, the decision-making process of each agent, as well as the e-procurement negotiation process are supported by argumentation. In the concrete use case, Al, as a requester, uses argumentation to collect information on the available services and on the suppliers. For instance, Al can ask Alice its opinion about Bob and ask to justify it (by providing an argument for it). Al (respectively Bob), as a requester (respectively as a provider), uses argumentation to decide which service it needs (respectively it can provide) taking into account the conflicting preferences/constraints and possibly the inconsistency of information it has gathered. Moreover, through argumentation, the participants provide an interactive and intelligible explanation of their choices. For instance, Al can argue that a service is a good deal when its cost is low. The previous argument will incite Bob to suggest services with a low cost to reach quickly a good deal. Thus, in our framework agents can use argumentation to influence each other.

3 Background on Argumentation and Protocol Language

In this section we discuss about the background formalisms that are necessary to understand the interaction happening in our ArguGRID e-procurement scenario.

3.1 MARGO

MARGO[6] (Multiattribute ARGumentation framework for Opinion explanation), written in Prolog, is the engine developed in the ArguGRID project for service selection and partner selection. We briefly present here the computational argumentation framework for decision-making which has been proposed in [45]. We do not describe the computational counterpart of MARGO (see [46] for more details) but the reader can notice that our framework is computed by the dialectical proof procedure of [19] extended in [23]. Concretely, MARGO is built upon the argumentation engine CaSAPI[7]. So that, we can compute the decisions. Additionally, MARGO models the intuition that high-ranked goals are preferred to low-ranked goals which can be withdrawn. At first, we introduce here argumentation. Then, we define the framework which captures decision making problems. Finally, we define the arguments and their interaction.

3.1.1 Abstract Argumentation

The framework proposed in this chapter is based on Dung's abstract approach to defeasible argumentation [18] which considers *arguments* as atomic and abstract entities interacting through a *defeat* relation over these[8].

Definition 1 (AAF). An *abstract argumentation framework* AAF is a pair $\langle \mathscr{A}, \text{ defeats } \rangle$ where $\mathscr{A}$ is a finite set of arguments and defeats $\subseteq \mathscr{A} \times \mathscr{A}$ is a binary relation over $\mathscr{A}$. We say that an argument $\mathtt{b}$ defeats an argument $\mathtt{a}$ if $(\mathtt{b},\mathtt{a}) \in$ defeats . Moreover, we say that a set of arguments $\mathtt{S}$ defeats an argument $\mathtt{a}$ if $(\mathtt{b},\mathtt{a}) \in$ defeats for some $\mathtt{b}$ in $\mathtt{S}$.

An argument can be viewed as a reason supporting a claim which can be disputed by other reasons defeating it.

According to this framework, Dung introduces various extension-based semantics in order to analyse whenever a set of arguments can be considered as collectively justified.

Definition 2 (Semantics). Let $\langle \mathscr{A}, \text{ defeats } \rangle$ be an AAF. For a set of arguments $\mathtt{S} \subseteq \mathscr{A}$, we say that:

- $\mathtt{S}$ is *conflict-free* iff $\forall \mathtt{a},\mathtt{b} \in S$ $\mathtt{a}$ does not defeat $\mathtt{b}$;
- $\mathtt{S}$ is *admissible* iff $\mathtt{S}$ is conflict-free and $\mathtt{S}$ defeats every argument $\mathtt{a}$ such that $\mathtt{a}$ defeats some arguments in $\mathtt{S}$;
- $\mathtt{S}$ is *preferred* iff $\mathtt{S}$ is maximally (wrt set inclusion) admissible;
- $\mathtt{S}$ is *complete* iff $\mathtt{S}$ is admissible and $\mathtt{S}$ contains all arguments $\mathtt{a}$ such that $\mathtt{S}$ defeats all defeaters against $\mathtt{a}$;
- $\mathtt{S}$ is *grounded* iff $\mathtt{S}$ is minimally complete.

[6] http://margo.sourceforge.net

[7] http://www.doc.ic.ac.uk/ ft/CaSAPI/

[8] Actually, the defeat relation is called attack in [18].

These declarative *semantics* capture various degrees of justification ranging from very permissive conditions, called *credulous*, to restrictive requirements, called *sceptical*. The semantics of an admissible (or preferred) set of arguments is credulous. However, there might be several conflicting admissible sets. That is the reason why various sceptical semantics have been proposed, notably the grounded semantics. Since an ultimate choice between various justified sets of alternatives is not always possible, we consider in this chapter only the credulous semantics.

3.1.2 Decision Framework

The problem of selecting services and partners can be seen as a multi-criteria decision problem with incomplete knowledge. In this chapter, we use influence diagrams to create a model and a representation of this kind of problem.

We assume that users provide, via a Graphical User Interface (GUI), influence diagrams. *Influence diagrams* are simple graphical representations of decision problems [13]. The elements of decision problems (decisions to make, uncertain events, and the value of outcomes) are represented in influence diagrams as nodes of different shapes. These nodes are linked to show the relationship between the elements. The nodes are of the following types: *decision nodes* (represented as squares), *chance nodes* (represented as ovals), and *value nodes* (represented as rectangles with rounded corners). Nodes are connected in a graph connected by arrows, called *arcs*. We call a node at the beginning of an arc a *predecessor* and one at the end of an arc a *successor*. The nodes are connected by arcs where predecessors are independent and affect successors. Influence diagrams which are properly constructed have no cycles. In order to capture multi-criteria decision making, it is convenient to include an additional type of node that aggregates results from predecessor nodes. An *abstract value* node is a special kind of value node represented by a rectangle with rounded corners and double borders. A *concrete value* is specified for every possible combination of decisions and events that feed into this node. By contrast, an abstract value is specified for every possible combination of values that feed into this node. In many cases, decision is a matter of trade-offs between the attributes of the outcomes. In such a case, it is possible to represent explicitly the multiple attributes with a hierarchy of values where the top, abstract values aggregate the lower, concrete values. In addition, the GUI allows the user to communicate user-specific preferences over values and events.

Influence diagrams can be mapped into decision frameworks of the following form:

Definition 3 (Decision framework). A *decision framework* is a tuple $\mathscr{D} = \langle \mathscr{L}, \mathscr{A}sm, \mathscr{I}, \mathscr{T}, \succ \rangle$, where:

- $\mathscr{L}$ is the *object language* which captures the statements about the decision problem;
- $\mathscr{A}sm$, is a set of sentences in $\mathscr{L}$ which are taken for granted, called *assumptions*;
- $\mathscr{I}$ is the *incompatibility relation*, i.e. a binary relation over atomic formulas which is asymmetric;

- $\mathscr{T}$ is the *theory* which gathers the statements;
- $\succ \subseteq \mathscr{T} \times \mathscr{T}$ is a transitive, irreflexive and asymmetric relation over $\mathscr{T}$, called the *priority* relation.

In the object language $\mathscr{L}$, we distinguish six disjoint components:

- a set of *abstract goals*, i.e. some propositional symbols which represent the abstract features that the decisions must exhibit implicitly;
- a set of *concrete goals*, i.e. some propositional symbols which represent the concrete features that the decisions must exhibit explicitly;
- a set of *decisions*, i.e. some predicate symbols which represent the actions which must be performed or not;
- a set of *alternatives*, i.e. some constant symbols which represent the mutually exclusive actions for each decision;
- a set of *beliefs*, i.e. some predicate symbols which represent epistemic statements;
- a set of *names* of rules in $\mathscr{T}$ (each rule has a distinguished name).

The abstract (respectively concrete) goals represent the abstract (respectively concrete) value nodes, the decisions represent the decision nodes, and the beliefs represent the chance nodes in influence diagrams. Since we consider multi-criteria decision problems, goals are structured hierarchically, where the top, abstract goals aggregate the independent lower goals.

We explicitly distinguish *assumable* (respectively *non-assumable*) literals which can (respectively cannot) be taken for granted, meaning that they can (respectively cannot) be assumed to hold as long as there is no evidence to the contrary. Decisions as well as some beliefs can be taken for granted. In this way, a decision framework can capture incomplete knowledge.

Since we want to consider conflicts in this object language, we need some form of negation. For this purpose, we consider *strong negation*, also called *explicit* or *classical negation*, and *weak negation*, also called *negation as failure*. A strong literal is an atomic first-order formula, possible preceded by strong negation $\neg$. A weak literal is a literal of the form $\sim L$, where L is a strong literal of $\mathscr{L}$. $\neg L$ says "L is definitely not the case", while $\sim L$ says "There is no evidence that L is the case". In order to express the mutual exclusion between statements, such as the different alternatives for a decision, we define the *incompatibility relation* (denoted by $\mathscr{I}$) as a binary relation over atomic formulas which is asymmetric. In other words, the incompatibility relation captures the conflicts between decisions, beliefs and goals. Whatever the atom L is, we have $L \mathscr{I} \neg L$ and $\neg L \mathscr{I} L$, while we have $L \mathscr{I} \sim L$ but we do not have $\sim L \mathscr{I} L$. Obviously, $D(a_1) \mathscr{I} D(a_2)$ and $D(a_2) \mathscr{I} D(a_1)$, D being a decision predicate, a_1 and a_2 being different[9] alternatives for D. We say that two sets of sentences Φ_1 and Φ_2 are incompatible (denoted $\Phi_1 \mathscr{I} \Phi_2$) iff there is a sentence ϕ_1 in Φ_1 and a sentence ϕ_2 in Φ_2 such that $\phi_1 \mathscr{I} \phi_2$.

A theory gathers the statements about the decision problem.

[9] Notice that in general a decision can be addressed by more than two alternatives.

Definition 4 (Theory). A *theory* $\mathscr{T}$ is an extended logic program, *i.e* a finite set of rules R: $L_0 \leftarrow L_1,\ldots,L_j,\sim L_{j+1},\ldots,\sim L_n$ with $n \geq 0$, each L_i (with $i \geq 0$) being a strong literal in $\mathscr{L}$. The literal L_0, called the *head* of the rule, is denoted $\texttt{head}(R)$. The finite set $\{L_1,\ldots,\sim L_n\}$, called the *body* of the rule, is denoted $\texttt{body}(R)$. The body of a rule can be empty. In this case, the rule, called a *fact*, is an unconditional statement. R, called the unique *name* of the rule, is an atomic formula of $\mathscr{L}$. All variables occurring in a rule are implicitly universally quantified over the whole rule. A rule with variables is a scheme standing for all its ground instances.

For simplicity, we will assume that the names of rules are neither in the bodies nor in the head of the rules thus avoiding self-reference problems. Since we obtain our decision problems from influence diagrams, we assume that the elements in the body of rules are independent, the decisions do not influence the beliefs, and the decisions have no side effects.

Considering a decision problem, we distinguish:

- *goal rules* of the form R: $G_0 \leftarrow G_1,\ldots,G_n$ with $n > 0$. Each G_i is a goal literal in $\mathscr{L}$. The head of the rule is an abstract goal (or its strong negation). According to this rule, the abstract goal is promoted (or demoted) by the combination of goal literals in the body;
- *epistemic rules* of the form
  R: $B_0 \leftarrow B_1,\ldots,B_n$ with $n \geq 0$. Each B_i is a belief literal of $\mathscr{L}$. According to this rule, B_0 is true if the conditions $B_1,\ldots,B_n$ are satisfied;
- *decision rules* of the form
  R: $G \leftarrow D(a),B_1,\ldots,B_n$ with $n \geq 0$. The head of the rule is a concrete goal (or its strong negation). The body includes a decision literal ($D(a) \in \mathscr{L}$) and a set of belief literals possibly empty. According to this rule, the concrete goal is promoted (or demoted) by the decision $D(a)$, provided that conditions $B_1,\ldots,B_n$ are satisfied.

Considering statements in the theory is not sufficient to make a decision. In order to evaluate the previous statements, other relevant pieces of information should be taken into account, such as the uncertainty of beliefs, the priority between goals, or the expected utilities of the decisions. For this purpose, we consider the *priority* relation $\succ$ on the rules in $\mathscr{T}$, which is transitive, irreflexive and asymmetric. $R_1 \succ R_2$ can be read "R_1 has priority over R_2". $R_1 \not\succ R_2$ can be read "R_1 has no priority over R_2", either because R_1 and R_2 are *ex æquo*, or because R_1 and R_2 are not comparable.

In this work, we consider that all rules are potentially defeasible and that the priorities are domain-specific and extra-logical features, neither determined nor justified by consideration of logic. The priority over concurrent rules depends of the nature of rules. Rules are *concurrent* if their heads are identical or incompatible. We define three priority relations between concurrent rules:

- the priority over *decision rules* comes from the *expected utility* of decisions. The priority of such rules corresponds to the expectation of the conditional decision in reaching the goal literal. For instance, we need to choose a train or a flight in order to go from Pisa to Roma. Even if a direct flight exists, train is cheaper.

The expected utilities of these decisions can be captured by the priority between concurrent decision rules;

- the priority over *goal rules* comes from the *priority* overs goals. The priority of such rules corresponds to the relative importance of the combination of (sub)goals in the body as far as reaching the goal literal in the head is concerned. For instance, we prefer a cheap travel rather than an expensive one which is fast. This preference can be captured by the priority between concurrent goal rules;
- the priority over *epistemic rules* comes from the *uncertainty* of knowledge. The priority of such rules corresponds to the likelihood of the rules. For instance, an Italian travel agency asserts that there is a direct flight between Pisa and Roma, while an international travel agency asserts that this is not the case. If you trust more the Italian agency, the fact that a direct flight exists is likely. The likelihood of a belief can be captured by the priority between concurrent epistemic rules.

3.1.3 Structure of Arguments

In order to turn the decision framework presented in the previous section into a concrete argumentation framework, we need first to define the notion of argument. Since we want that our AF not only suggests some decisions but also provides an intelligible explanation of them, we adopt a tree-like structure of arguments. We adopt here the tree-like structure for arguments proposed in [62] and we extend it with presumptions on the missing information.

Informally, an argument is a deduction for a conclusion from a set of suppositions represented as a tree, with conclusion at the root and suppositions at the leaves. Nodes in this tree are connected by the inference rules, with sentences matching the head of an inference rule connected as parent nodes to sentences matching the body of the inference rule as children nodes. The leaves are either suppositions or the special extra-logical symbol θ, standing for an empty set of premises. Formally:

Definition 5 (Argument). An *argument* is composed of a conclusion, a top rule, some premises, some suppositions, and some sentences. These elements are abbreviated by the corresponding prefixes. An argument $\mathtt{a}$ may be one of the following:

1. a *hypothetical argument* with:

$$\begin{aligned}&\mathtt{conc}(\mathtt{a}) = L,\\ &\mathtt{top}(\mathtt{a}) = \theta,\\ &\mathtt{premise}(\mathtt{a}) = \emptyset,\\ &\mathtt{supp}(\mathtt{a}) = \{L\},\\ &\mathtt{sent}(\mathtt{a}) = \{L\}.\end{aligned}$$

   where L is an assumable belief literal.
   or
2. a *built argument* which may be

   2.1) a *trivial* argument $\mathtt{a}$ built upon a fact f in $\mathscr{T}$ (i.e. $\mathtt{body}(f) = \emptyset$), defined as follows:

$$\begin{aligned}&\texttt{conc}(\texttt{a}) = \texttt{head}(f),\\ &\texttt{top}(\texttt{a}) = f,\\ &\texttt{premise}(\texttt{a}) = \emptyset,\\ &\texttt{supp}(\texttt{a}) = \emptyset,\\ &\texttt{sent}(\texttt{a}) = \{\texttt{head}(f)\}.\end{aligned}$$

2.2) a *tree* argument a built upon the rule r and the set $\{\texttt{a}_1,\ldots,\texttt{a}_n\}$ of arguments, where r is a rule in $\mathscr{T}$ with $\texttt{body}(r) = \{L_1,\ldots,L_j,\sim L_{j+1},\ldots,\sim L_n\}$ and there is a collection of arguments $\{\texttt{a}_1,\ldots,\texttt{a}_n\}$ such that, for each strong literal $L_i \in \texttt{body}(r)$, $\texttt{conc}(\texttt{a}_i) = L_i$ with $i \leq j$ and for each weak literal $\sim L_i \in \texttt{body}(r)$, $\texttt{conc}(\texttt{a}_i) = \sim L_i$ with $i > j$. a is defined as follows:

$$\begin{aligned}&\texttt{conc}(\texttt{a}) = \texttt{head}(r),\\ &\texttt{top}(\texttt{a}) = r,\\ &\texttt{premise}(\texttt{a}) = \texttt{body}(r),\\ &\texttt{supp}(\texttt{a}) = \cup_{\texttt{a}_i\in\{\texttt{a}_1,\ldots,\texttt{a}_n\}}\texttt{supp}(\texttt{a}_i),\\ &\texttt{sent}(\texttt{a}) = \{\texttt{head}(r)\}\cup\texttt{body}(r)\\ &\qquad\qquad \cup_{\texttt{a}_i\in\{\texttt{a}_1,\ldots,\texttt{a}_n\}}\texttt{sent}(\texttt{a}_i).\end{aligned}$$

The set of arguments $\{\texttt{a}_1,\ldots,\texttt{a}_n\}$ is called the set of *subarguments* of a (denoted $\texttt{sbarg}(\texttt{a})$).

The top rule of an argument is the rule whose head is the conclusion of the argument. Notice that the subarguments of a tree argument concluding the weak literals in the body of the top rule are hypothetical arguments. Indeed, the conclusion of a hypothetical argument could be a strong or a weak literal while the conclusion of a built argument is a strong literal. As in [62], we consider composite arguments, called *tree* arguments, and atomic arguments, called *trivial* arguments. Contrary to other definitions of arguments (set of assumptions, set of rules), our definition considers that the different premises can be challenged and can be supported by subarguments. In this way, arguments are intelligible explanations. Moreover, we consider *hypothetical* arguments which are built upon missing information or a suggestion, i.e. a decision. In this way, our framework allows to reason further by making suppositions related to unknown beliefs and over possible decisions.

3.1.4 Interaction

The interactions between arguments may come from the incompatibility of their sentences, from their nature (hypothetical or built) and from the priority over rules. We examine in turn these different sources of interaction.

Since their sentences are conflicting, the structured arguments interact with one another. For this purpose, we define the following attack relation.

Definition 6 (Attack relation). Let a and b be two arguments. a *attacks* b iff $\texttt{sent}(\texttt{a})\ \mathscr{I}\ \texttt{sent}(\texttt{b})$.

According to this definition, if an argument attacks a subargument, the whole argument is attacked.

Since arguments are more or less hypothetical, we define the size of their suppositions.

Definition 7 (Supposition size). Let $\mathtt{a}$ be an argument. The *size of suppositions* for $\mathtt{a}$, denoted $\mathtt{suppsz(a)}$, is the number of supposition of $\mathtt{a}$: $\mathtt{suppsz(a)} = |\mathtt{supp(a)}|$.

Namely, the size of suppositions for an argument is the number of decision literals and assumable belief literals in the sentences of the argument.

Since arguments have different natures (hypothetical or built) and the top rules of built arguments are more or less strong, we define the strength relation as follows.

Definition 8 (Strength relation). Let $\mathtt{a}_1, \mathtt{a}_2$ be two built arguments. $\mathtt{a}_1$ is stronger than $\mathtt{a}_2$ (denoted $\mathtt{a}_1 \succ \mathtt{a}_2$) iff

1. either $(\mathtt{top(a_1)} \succ \mathtt{top(a_2)})$, then $\mathtt{a}_1 \succ \mathtt{a}_2$;
2. or $(\mathtt{top(a_1)} \not\succ \mathtt{top(a_2)}) \wedge (\mathtt{suppsz(a_1)} < \mathtt{suppsz(a_2)})$, then $\mathtt{a}_1 \succ \mathtt{a}_2$.

An argument is stronger than another argument if the top rule of the first argument has a proper higher priority that the top rule of the second argument, or if it is not the case but the number of suppositions made in the first argument is properly smaller than the number of suppositions made in the second argument.

The two previous relations can be combined.

Definition 9 (Defeats). Let $\mathtt{a}$ and $\mathtt{b}$ be two arguments. $\mathtt{a}$ *defeats* $\mathtt{b}$ iff: i) $\mathtt{a}$ attacks $\mathtt{b}$ and ; ii) $\neg(\mathtt{b} \succ \mathtt{a})$.

Our notion of argument and this defeat relation can be used within the Dung's seminal calculus of opposition.

The implementation of this argumentation framework, called MARGO[10], is built upon the implementation of [20] in the CaSAPI system [23].

3.2 Protocol Language

In the ArguGRID project, we use the Lightweight Coordination Calculus (LCC) of [55] which is a declarative logic programming language in the style of Prolog, augmented with CCS (a process calculus for communicating systems). LCC allows to drive all social interactions and allows to write once-execute everywhere protocols.

Figure 4 defines the syntax of the LCC protocol language. A protocol consists of a set of agent clauses, $A^{\{n\}}$. An agent clause is the series of communicative actions expected to be performed by an agent adopting the role defined by the agent definition. This agent definition consists of a role (*role*) and unique identifier (*ag*). The roles act as a bounding box for a set of states and transitions. The agent definition is expanded by a number of operations. Operations can be classified in three ways: actions, control flow, and conditionals. Actions are the sending or receiving of messages, a no op, or the adoption of a role. Control flow operations temporally order

[10] http://margo.sourceforge.net

$\mathscr{P}$	∈ Protocol	$::= A^{\{n\}}$
$\mathscr{A}$	∈ AgentClause	$::= \lambda$ :: op.
λ	∈ AgentDefinition	agent(*role*,*ag*)
op	∈ Operation	no op
		λ
	(Precedence)	—(op)
	(Send)	$M \Rightarrow \lambda$
	(Receive)	$M \Leftarrow \lambda$
	(Sequence)	—op1 then op2
	(Parallelization)	—op1 par op2
	(Choice)	—op1 or op2
	(Prerequiste)	$(M \Rightarrow \lambda) \leftarrow \psi$
	(Consequence)	$\psi \leftarrow (M \Rightarrow \lambda)$
M	∈ message	$::= \langle m, \mathscr{P} \rangle$

Fig. 4 An Abstract Language of the Protocol Language

the individual actions. Actions can be sequentially ordered, performed simultaneously without regard to order, or given a choice point. The definition of the double arrows denote messages M being sent and received. On the left hand side of the double arrow is the message and on the right-hand side is the other agent involved in the interaction.

Constraints can fortify or clarify the semantics of the protocols. Those occurring on the left of the $\leftarrow$ are post-conditions and those occurring on the right are preconditions. The symbol ψ represents a first order proposition. For example, an agent receiving a protocol with the constraint to believe a proposition $\mathtt{s}$ upon being informed of $\mathtt{s}$ can infer that the agent sending the protocol has a particular semantic interpretation of the act of informing other agents of propositions. The operation $(M \Rightarrow \lambda) \leftarrow \psi$ is understood to mean that message M is being sent to the agent defined as λ on the condition that ψ is satisfiable. The operation $\psi \leftarrow (M \Rightarrow \lambda)$ means that once M is received from agent, ψ holds.

4 Agent Architecture

In this section, we outline the mind/body architecture of our agents, focusing on the mind component.

Our agent architecture, pictured in Figure 5, is adapted from the mind/body architecture of [9], extended in GOLEM [10]. The *body* senses what is external to it by using the *Communication Module* (CM), which can access the external world, i.e. the events generated by the *Graphical User Interface* (GUI) and messages coming from other agents as well as the registry used for the partner discovery. Messages received are then stored in the *Incoming Message Queue* (IMQ) until they are treated. Similarly, the messages that the agent wants to send are stored in the *Outgoing Message Queue* (OMQ). The mind and the body can function as co-routines, thus

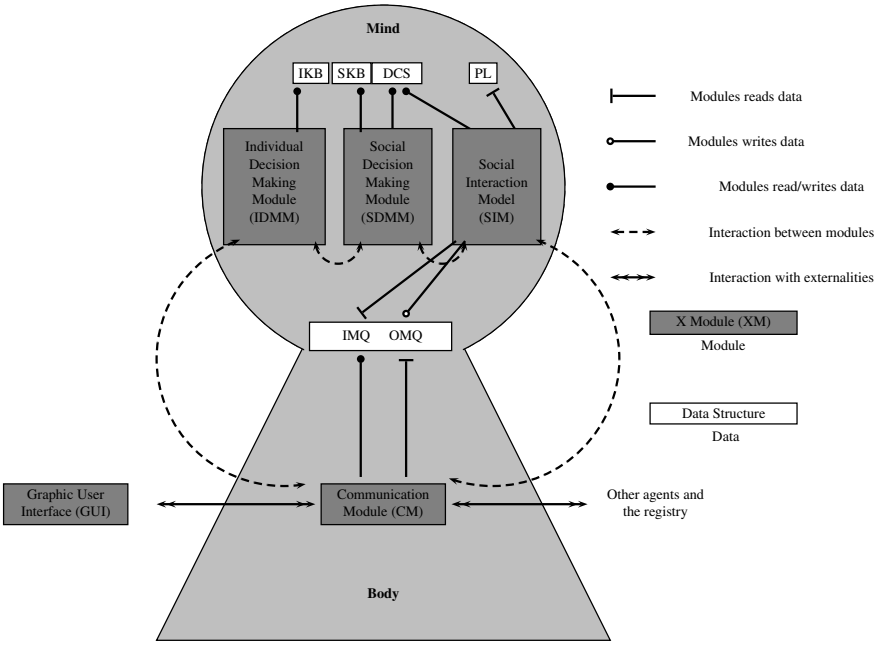

Fig. 5 The modular architecture of agents

allowing the reasoning processes of the mind to be performed concurrently with the body sending and receiving messages.

Our cognitive agents' mind is divided into three modules:the *Individual Decision Making Module* (IDMM), the *Social Decision M aking Module* (SDMM) and the the *Social Interaction Module* (SIM).

4.1 Individual Decision Making

The *Individual Decision Making Module* (IDMM), supports the reasoning about the provided and requested services. The IDMM is supported by the concrete data structures in the *Individual Knowledge Base* (IKB). Decisions are made according to the user's requirements or competencies about the services, the alternative types of services, and the users' preferences and constraints.

Users can provide their requirements to the agents through a GUI to draw influence diagrams, where they can display the structure of the decision problem about the services provided or to be obtained. In addition, the GUI allows the user to communicate user-specific details, in particular preferences and constraints. For instance, Figure 6 gives the influence diagram related to the evaluation of services by an A-type agent, e.g. `Al`. The top, main goal (`provision`) is split into independent abstract sub-goals concerning the cost (`cost`) and the quality of service (`qos`).

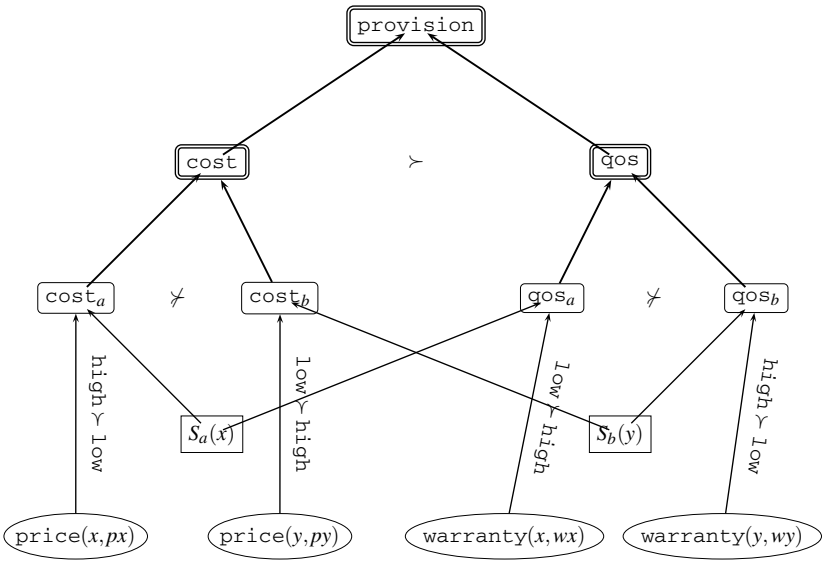

Fig. 6 Influence diagram to structure the decision

These sub-goals are reduced to further concrete sub-goals. For instance, the quality of service depends on the quality of the service S_a ($\texttt{qos}_a$) and on the quality of the service S_b ($\texttt{qos}_b$). While abstract goals just reflect the user's needs, concrete goals provide criteria to evaluate different alternatives.

The main goal, the provision of a composite service S, needs to be addressed by some decisions, e.g. on which concrete S_a and S_b service to adopt (by appropriately instantiating variables x and y [11] in Figure 6). Note that `Al` is a candidate provider of service S_a in our use case, but may be able to provide several instances of this service, namely x may be instantiated in many different ways. Instead, `Al` is not a candidate provider of service S_b, and needs to choose one. These decisions depend on the agent knowledge, namely information about the concrete services provided by `Al` itself and B-type providers (e.g. `price`, `warranty`).

The user provides also his preferences and constraints. For example, the user may specify that `cost` is more important to the user than `qos` as far as reaching `provision` is concerned. *Priorities* are attached to goals, decisions and knowledge in an influence diagram to represent the preferences over goals, the expected utilities of decisions, and the uncertainty of the knowledge.

The structure of the decision problem related to the evaluation of services and the associated priorities are stored within the agent's knowledge bases (in particular the IKB) and reasoned upon by the IDMM. In ArguGRID, the IDMM is

[11] Throughout the chapter we adopt the following convention: variables are in italics and constants are in typescript font.

realised using our argumentation framework for decision making. For instance, the knowledge bases corresponding to the influence diagram of Figure 7 are represented in Table 1. The goal rules are depicted at top, while the decision rules are depicted at the bottom. In this example, there is no epistemic rule. That is the reason why the incorporation of supposition on missing information is essential to perform the individual reasoning of requester agents. A rule above another one has priority over it. To simplify the graphical representation of the rules, they are stratified in non-overlapping subsets, i.e. different levels. The *ex æquo* rules are grouped in the same level. Non-comparable rules are arbitrarily assigned to a level. The goal rules express that achieving `cost` and `qos` is ideally required to reach `provision`, but this can be relaxed: achieving the goal `cost` is enough to reach `provision` ($\mathtt{r}_{012} \succ \mathtt{r}_{01}$). Contrary to the other rules in Table 1, $\mathtt{r}_{01}$ is not in the IKB which reflects the own user requirements represented by `Al` but $\mathtt{r}_{01}$ is the output of the previous interaction between `Al` and the buyer stored in the DCS. $\mathtt{r}_{01}$ reflects the preference of the customer represented by the buyer agent.

$\mathtt{r}_{012} : \mathtt{provision} \leftarrow \mathtt{cost}, \mathtt{qos}$
$\mathtt{r}_{01} : \mathtt{provision} \leftarrow \mathtt{cost}$
$\mathtt{r}_{134} : \mathtt{cost} \leftarrow \mathtt{cost}_a, \mathtt{cost}_b$
$\mathtt{r}_{256} : \mathtt{qos} \leftarrow \mathtt{qos}_a, \mathtt{qos}_b$

$\mathtt{r}_{32}(x) : \mathtt{cost}_a \leftarrow S_a(x), \mathtt{price}(x, \mathtt{high})$
$\mathtt{r}_{41}(y) : \mathtt{cost}_b \leftarrow S_b(y), \mathtt{price}(y, \mathtt{low})$
$\mathtt{r}_{51}(x) : \mathtt{qos}_a \leftarrow S_a(x), \mathtt{warranty}(x, \mathtt{low})$
$\mathtt{r}_{62}(y) : \mathtt{qos}_b \leftarrow S_b(y), \mathtt{warranty}(y, \mathtt{high})$
$\mathtt{r}_{31}(x) : \mathtt{cost}_a \leftarrow S_a(x), \mathtt{price}(x, \mathtt{low})$
$\mathtt{r}_{42}(y) : \mathtt{cost}_b \leftarrow S_b(y), \mathtt{price}(y, \mathtt{high})$
$\mathtt{r}_{52}(x) : \mathtt{qos}_a \leftarrow S_a(x), \mathtt{warranty}(x, \mathtt{high})$
$\mathtt{r}_{61}(y) : \mathtt{qos}_b \leftarrow S_b(y), \mathtt{warranty}(y, \mathtt{low})$

Table 1 The goal rules (at top) and the decision rules (at bottom) representing the users requirements

These concrete data structures (rules and priorities) provide the backbone of arguments. For instance, `Al` can build an admissible argument concluding that the goal related to the cost of the service S_b is reached by choosing $S_b(x)$ if we suppose that the price of service $S_b(x)$ is low.

The IDMM interacts with the other components of the architecture as follows. It interacts with the GUI, through the CM, which uses the GOLEM environment as a mediator to interact with other entities in the system and the user. The IDMM is informed and responds when a service is (or must be) instantiated. The IDMM interacts with the SDMM by asking or by providing the instantiation of the abstract

or partially instantiated service, and by being informed when the provision of a concrete service is (or must be) accomplished. In this way, the IDMM module shifts from the goals and the preferences provided by the user to an abstract representation of atomic services (or composite services, as appropriate). For instance, `Al` can build an admissible argument concluding that the goal related to the provision of the service S is reached if we suppose that the price of the service $S_b(y)$ is low. This is then turned into a concrete representation (choice of y fulfilling the constraint) by the SDMM.

4.2 Social Decision Making

The *Social Decision Making Module* (SDMM) reasons about the concrete instances of services that can be provided/requested. Decisions are made according to user's requirements or competencies, the knowledge about the potential partners and the alternative concrete services, and preferences over them. For this purpose the SDMM is supported by the concrete data structures in the *Social Knowledge Base* (SKB) and in the *Dialogical Commitment Store* (DCS).

The dialogical commitment store is an internal data structure which contains propositional and action suggestions involving the agent, namely with the agent being either the debtor or the creditor. This data structure is shared by the SIM and the SDMM. Concretely, the dialogical commitment store may contain the concrete representation of atomic or composite services and the representation of the partners exchanged during the dialogues, while the SKB contains the concrete representation of atomic or composite services provided by the agent. Moreover, the SKB contains preferences about the services and the partners. The selection (respectively the suggestion) of concrete services is made according to the user's requirements (respectively competencies) about the alternative concrete services, the information about the partners, and preferences over them.

Figure 7 represents the negotiation problem of S_b from an A-type agent's viewpoint, e.g `Al`. The evaluation of the contract (`good_deal`) depends on the provision of the service (`provision`) as considered by the IDMM and depends also on the supplier (`supplier`). The evaluation of the partners depends on their representation (`representation`) and on their performance (`performance`). The supplier's performance is influenced by knowledge about customer testimonials ($\texttt{testimonials}(x,v)$) depending on the number of previous collaborations with these suppliers ($\texttt{previous}(x,n)$) and the average satisfaction in these collaborations ($\texttt{satisfaction}(x,v)$). For simplicity, preferences are not depicted in Figure 7. In the project ArguGRID, the SDMM, like the IDMM, is built upon our argumentation framework for decision making. Statements and priorities are recorded within the agent's SKB, the agent's DCS, and reasoned upon by the SDMM. Alternatively, the user can directly fulfil the SKB and the DCS. The statements corresponding to the influence diagram of Figure 7 are represented in Table 2. The goal rules, the epistemic rules and the decision rules are depicted in the table 2. For instance, the agent whose influence diagram is given in Figure 6 deems the suppliers

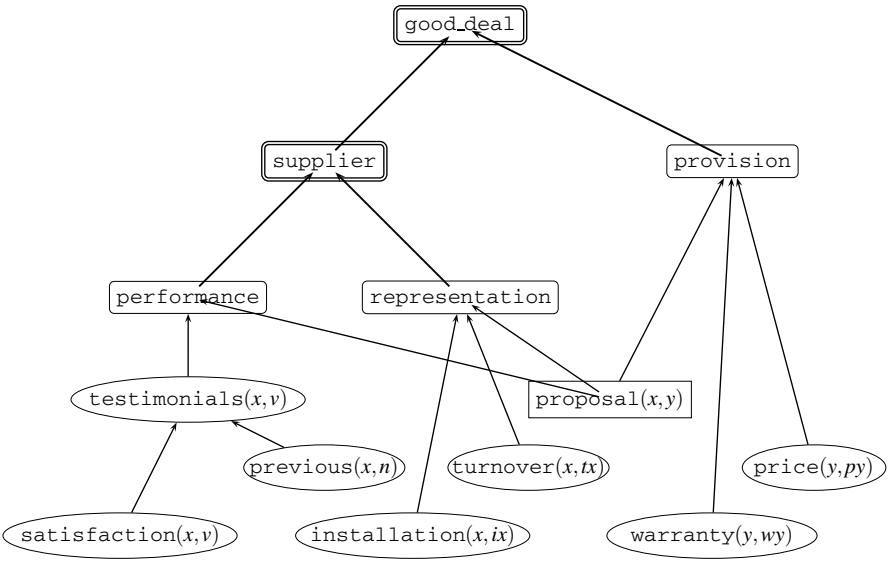

Fig. 7 Influence diagram to structure the negotiation

with an annual turnover greater than two million euros (`turnover`) and at least 50 installations (`installation`). For this purpose, the rules $\mathtt{r}_{31}(x,y)$ and $\mathtt{r}_{32}(x,y)$ are included in the SKB.

The dialogical commitment store of `Al`, which includes the public statements of agents which `Al` is aware of, include the suggestions involving `Al`: either `Al` is the creditor of the suggestion, for instance $\mathtt{commit}(\mathtt{Bob},[\mathtt{good\_deal},S_b(\mathtt{e}),\emptyset])$ is added to the dialogical commitment store when `Bob` suggests it (see the next section); or `Al` is the debtor of the suggestion, for instance $\mathtt{commit}(\mathtt{Al},[\mathtt{good\_deal},S_b(\mathtt{e}),\emptyset])$ is added to the dialogical commitment store when `Al` accepts them (see the next section).

The SDMM interacts with the IDMM by exchanging that abstract service which is (or must be) instantiated and by communicating when a concrete service is (or must be) set up. The SDMM interacts with the SIM (see the next section), by notifying it that the agent needs to play a certain role using a particular protocol, by being informed by the SIM when some offers, some proposals, and some arguments must be evaluated or built and by informing the SIM when the offers, the proposals, the arguments have been evaluated or built.

In this way, the SDMM reasons and takes decision about the proposals and arguments which are exchanged during the dialogues. For instance, `Al` can built an admissible argument concluding that the goal related to the cost of the service S_b is reached since the price of the service $S_b(\mathtt{c})$ is low. This argument is useful for `Al` to justify its choice, $S_b(\mathtt{c})$, in front of `Bob`.

Table 2 The goal theory (at top), the epistemic theory (at middle) and the decision theory (at bottom) corresponding to the social statements

$r_{012} : \texttt{good\_deal} \leftarrow \texttt{supplier}, \texttt{provision}$

$r_{01} : \texttt{good\_deal} \leftarrow \texttt{supplier}$
$r_{134} : \texttt{supplier} \leftarrow \texttt{performance}, \texttt{representation}$
$r_{256} : \texttt{provision} \leftarrow \texttt{cost}_b, \texttt{qos}_b$

$r_{25} : \texttt{provision} \leftarrow \texttt{cost}_b$

$r_{26} : \texttt{provision} \leftarrow \texttt{qos}_b$
$r_{02} : \texttt{good\_deal} \leftarrow \texttt{provision}$

$f_1 : \texttt{testimonials}(\texttt{Bob}, \texttt{high}) \leftarrow$
$f_2 : \texttt{turnover}(\texttt{Bob}, 5) \leftarrow$
$f_3 : \texttt{installation}(\texttt{Bob}, 100) \leftarrow$
$f_4 : \texttt{price}(\texttt{d}, \texttt{high}) \leftarrow$
$f_5 : \texttt{warranty}(\texttt{d}, \texttt{low}) \leftarrow$
$f_6 : \texttt{price}(\texttt{c}, \texttt{low}) \leftarrow$
$f_7 : \texttt{warranty}(\texttt{c}, \texttt{high}) \leftarrow$
$f_8 : \texttt{price}(\texttt{e}, \texttt{low}) \leftarrow$
$f_9 : \texttt{warranty}(\texttt{e}, \texttt{low}) \leftarrow$
$f_{10} : \texttt{price}(\texttt{f}, \texttt{high}) \leftarrow$
$f_{11} : \texttt{warranty}(\texttt{f}, \texttt{high}) \leftarrow$

$r_{21}(x,y) : \texttt{performance} \leftarrow \texttt{proposal}(x,y), \texttt{testimonials}(x, \texttt{high})$
$r_{31}(x,y) : \texttt{representation} \leftarrow \texttt{proposal}(x,y), \texttt{turnover}(x,tx), tx > 2\text{M euros}$
$r_{32}(x,y) : \texttt{representation} \leftarrow \texttt{proposal}(x,y), \texttt{installation}(x,ix), ix > 50$
$r_{51}(x) : \texttt{cost}_b \leftarrow \texttt{proposal}(x,y), \texttt{price}(y, \texttt{low})$
$r_{61}(x) : \texttt{qos}_b \leftarrow \texttt{proposal}(x,y), \texttt{warranty}(y, \texttt{high})$

4.3 Social Interaction

The *Social Interaction Module* (SIM) drives the communications and interactions by the adherence to protocols. These protocol are concrete data structures and are stored in the *Protocol Library* (PL).

Decisions required to conduct the interaction are provided by the SDMM. In practice, the SDMM, once it has reasoned about the services requirements and from whom these services can be requested or to whom these services can be proposed, uses a boot strap mechanism that initiates the required protocol, the role the agent will play in that protocol, and the other participants. From this, the protocol engine in the SIM determines the appropriate message to be sent given those parameters. In cases where there is no ambiguity with respect to the message to be sent, the SIM will automatically send the message to the outgoing message queue from which the agent body will handle the transportation of the message to the recipients' incoming message queues. Where there is a decision to be made either between the choice of two locutions (e.g. whether to accept or reject an offer) to be sent or the instantiation of the content of the locution (e.g. the definition of a proposal), the SIM uses a

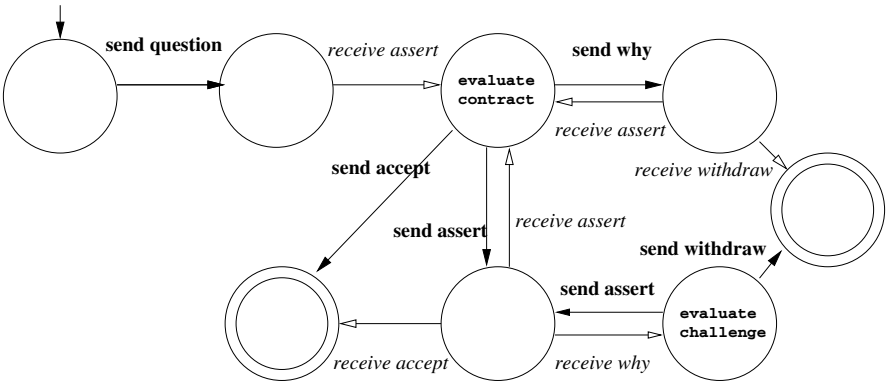

Fig. 8 Negotiation protocol for the requester

precondition mechanism to prompt the SDMM for a solution. Concretely, MARGO interfaces with LCC through the condition mechanism of utterances for a move. Upon the satisfaction of the precondition, the SIM sends the locution to the outgoing message queue. If it is necessary to update the dialogical commitment store of the agent, this can be done with the post condition mechanism which operates in a similar manner.

The agents utter messages to exchange goals, decisions, and knowledge. The syntax of messages is in conformance with a common communication language. We assume that each message:

- has an identifier, M_k;
- is uttered by a speaker (S_k);
- is addressed to a hearer (H_k);
- eventually responds to a message with identifier R_k
- is characterised by a speech act A_k composed of a locution and a content.

In our scenarios, the locution is one of the following: `question`, `assert`, `accept`, `why`, `withdraw` (see Table 3 below for examples). The content is a triple consisting of: a goal G_k, a decision D_k, and knowledge K_k. We will use θ to denote that no goal is given and $\emptyset$ to denote that no knowledge is provided.

In our scenario our agents use two protocols for two types of dialogue: information-seeking and negotiation (see Table 2). Figure 8 depicted our negotiation protocol from the requester viewpoint with the help of a deterministic finite-state automaton. The SDMM interfaces with the SIM's protocol through the condition mechanism to further elucidate the semantics of the protocol being used. For example, at one point in the dialogue the requester is able to send `accept`, `assert` and `why`. The choice of which locution to send is dependant on the SDMM being able to satisfy its precondition.

In the ArguGRID project, the SIM uses LCC to drive all social interactions. Figure 9 represents the set of clauses for the agents in our argumentation-based negotiation protocol. The role being used are:

- the requestor of the service (requestor(g_0, c, K)), g_0 being the main goal to reach, c being the contract, and K being the knowledge. c and K must be instantiated;
- the provider of the service (provider(g_0, c, K)), g_0 being the main goal to reach, c being the contract, and K being the knowledge. c and K must be instantiated;
- the evaluator of the proposal (evaluator(g_0, g_1, c_1, K_1)), g_0 (respectively g_1) being the main (respectively current) goal which is discussed, c_1 being the contract, and K_1 being the knowledge. c_1 and K_1 are related to the proposal;
- the sender of the proposal (proponent(g_0, g_1, c_1, K_1)), g_0 (respectively g_1) being the main (respectively current) goal which is discussed, c_1 being the contract, and K_1 being the knowledge. c_1 and K_1 are related to the proposal.

According to the first clause a(requestor(g_0, c, K), ag_1), ag_1 sends a question to the provider which submits a proposal. The clause for the provider role is the complement of the requestor's one. The second clause a(evaluator(g_0, g_1, c_1, K_1), ag_1) handles the next stage of the argumentation-based negotiation protocol, i.e. the acceptance, the counter-proposal, or the challenge of the proposal. A proposal is accepted, and so recorded in the dialogical commitment store[12], if it is supported by an admissible argument of the SDMM. Indeed, the condition *evaluate_contract*(g_0, c_1, K_1) is satisfied if there is an admissible argument of the SDMM such that the knowledge K_1 and the contract c_1 are in the sentences of the argument and g_0 is the conclusion of the argument. If it is not the case and another proposal, which was not yet suggested, is supported by an admissible argument of the SDMM, then the counter-proposal is asserted, and so recorded in the dialogical commitment store. Otherwise the motivation of the previous proposal is challenged. The clause for the proponent role is the complement of the evaluator's one. An argument which is challenged must be supported by a subargument. Indeed, the condition *evaluate_challenge*(g_1, g_2, c_1, K_1, K_2) is satisfied if: i) there is an admissible argument of the SDMM such that the goal g_2, the knowledge K_1, and the contract c_1 are in the sentences of the argument and g_1 is the conclusion of the argument; ii) there is an admissible argument of the SDMM such that the knowledge K_2 and the contract c_1 are in the sentences of the argument concluding g_2.

The top of the table 3 shows the speech acts exchanged between Al and Bob playing an information-seeking dialogue. This dialogue occurs at the step 4 of the e-procurement process involving Al and Bob (cf Figure 2). Following the protocol for this type of dialogue, the first move is for Al to pose a question to Bob, M_0. This locution seeks the price range for the available services S_b. Bob informs Al with several locutions providing the various services available and their price ranges (M_1, ..., M_4). A similar information-seeking dialogue is played between Bob and Al to inform the latter about the warranty ranges of the available services S_b. According

[12] Actually, the acceptance of a proposal creates an extra-dialogical commitments, i.e. one agent is obligated to provide a service and another is obligated to pay for it. Whether we ignore the task of enforcement, we still make enforcement possible.

a(requestor(g_0,*c*,*K*),ag_1)::=
question(g_0, c, K) $\Rightarrow$ *a*(*provider*(g_0, c, K),ag_2) **then**
commit(ag_2,[g_0,c_1,K_1]) $\leftarrow$ (*assert*(g_0, c_1, K_1) $\Leftarrow$ *a*(*provider*(g_0, c, K),ag_2)) **then**
a(*evaluator*(g_0, g_0, c_1, K_1),ag_2).

a(evaluator(g_0,g_1,c_1,K_1),ag_1)::=
(*accept*(g_0, c_1, K_1) $\Rightarrow$ *a*(*proponent*(g_0, g_1, c_1, K_1),ag_2)) $\leftarrow$
(*evaluate_contract*(g_0, c_1, K_1)) **and** commit(ag_1,[g_0,c_1,K_1])
or
(*assert*(g_0, c_2, K_2) $\Rightarrow$ *a*(*proponent*(g_0, g_1, c_1, K_1),ag_2)) $\leftarrow$
(*evaluate_contract*(g_0, c_2, K_1) **and not**(commit(ag_1,[g_0,c_2,K_2]))) **and**
commit(ag_1,[g_0,c_2,K_2]) **then**
a(*proponent*(g_0, g_1, c_2, K_2),ag_1))
or
(*why*(g_1, c_1, K_1) $\Rightarrow$ *a*(*proponent*(g_0, g_1, c_1, K_1),ag_2) **then**
commit(ag_2,[g_2,c_1,K_2]) $\leftarrow$ (*assert*(g_2, c_1, K_2) $\Leftarrow$ *a*(*proponent*(g_0, g_1, c_1, K_1),ag_2))
and *a*(*evaluator*(g_0, g_2, c_1, K_2),ag_1)) **or**
withdraw(g_1, c_1, K_1) $\Leftarrow$ *a*(*proponent*(g_0, g_1, c_1, K_1),ag_2)).

a(provider(g_0,*c*,*K*),ag_2)::=
question(g_0, c, K) $\Leftarrow$ *a*(*requestor*(g_0, c, K),ag_1) **then**
(*assert*(g_0, c_1, K_1) $\Rightarrow$ *a*(*requestor*(g_0, c, K),ag_1)) $\leftarrow$
(*evaluate_contract*(g_0, c_1, K_1)) **and**
commit(ag_2,[g_0,c_1,K_1]) **then**
a(*proponent*(g_0, g_0, c_1, K_1),ag_2).

a(proponent(g_0,g_1,c_1,K_1),ag_2)::=
commit(ag_1,[g_0,c_1,K_1]) $\leftarrow$ (*accept*(g_0, c_1, K_1) $\Leftarrow$ *a*(*evaluator*(g_0, g_1, c_1, K_1),ag_1))
or
commit(ag_1,[g_0,c_2,K_2]) $\leftarrow$ (*assert*(g_0, c_2, K_2) $\Leftarrow$ *a*(*evaluator*(g_0, g_1, c_1, K_1),ag_1))
then
a(*evaluator*(g_0, g_1, c_2, K_2),ag_2)
or
(*why*(g_1, c_1, K_1) $\Leftarrow$ *a*(*evaluator*(g_0, g_1, c_1, K_1),ag_1) **then**
(*assert*(g_2, c_1, K_2) $\Rightarrow$ *a*(*evaluator*(g_0, g_1, c_1, K_1),ag_1) $\leftarrow$
(*evaluate_challenge*(g_1, g_2, c_1, K_1, K_2)) **and**
commit(ag_2,[g_2,c_1,K_2]) **then**
a(*proponent*(g_0, g_2, c_1, K_2),ag_2)) **or**
withdraw(g_1, c_1, K_1) $\Rightarrow$ *a*(*evaluator*(g_0, g_1, c_1, K_1),ag_1)).

Fig. 9 Representation of the argumentation-based negotiation protocol

Table 3 Information seeking dialogue (top and middle) and negotiation dialogue (bottom)

M_k	S_k	H_k	A_k	R_k
M_0	Al	Bob	question(θ,$S_b(x)$,[price$_b$(x,px)])	θ
M_1	Bob	Al	assert(θ,S_b(c),[price$_b$(c,pc),low $\leq pc \leq$ medium])	M_0
M_2	Bob	Al	assert(θ,S_b(e),[price$_b$(e,pe),low $\leq pe \leq$ medium])	M_0
M_3	Bob	Al	assert(θ,S_b(d),[price$_b$(d,pd),medium $\leq pd \leq$ high])	M_0
M_4	Bob	Al	assert(θ,S_b(f),[price$_b$(f,pf),medium $\leq pf \leq$ high])	M_0

M_k	S_k	H_k	A_k	R_k
M_0	Al	Alice	question(performance,proposal(Bob,y),[testimonials(Bob,z)])	θ
M_1	Alice	Al	assert(performance,proposal(Bob,y),[testimonials(Bob,high)])	M_0
M_2	Al	Alice	why(θ,proposal(Bob,y),[testimonials(Bob,high)])	M_1
M_3	Alice	Al	assert(θ,proposal(Bob,y),[previous(Bob,10),satisfaction(Bob,high)])	M_2

M_k	S_k	H_k	A_k	R_k
M_0	Al	Bob	question(good_deal,$\langle cid$,Bob,Al,$S_b(x)$,[price(x,px),warranty(x,wx)]$\rangle$,$\emptyset$)	θ
M_1	Bob	Al	assert(good_deal,$\langle$c1,Bob,Al,$S_b(d)$,[price(d,high),warranty(d,low)]$\rangle$,$\emptyset$)	M_0
M_2	Al	Bob	assert(good_deal,$\langle$c2,Bob,Al,S_b(c),[price(c,low),warranty(c,high)]$\rangle$,$\emptyset$)	M_1
M_3	Bob	Al	why(good_deal,$\langle$c2,Bob,Al,S_b(c),[price(c,low),warranty(c,high)]$\rangle$,$\emptyset$)	M_2
M_4	Al	Bob	assert(cost$_{\text{Al}}$,$\langle$c2,Bob,Al,S_b(c),[price(c,low),warranty(c,high)]$\rangle$,$\emptyset$)	M_3
M_5	Bob	Al	assert(good_deal,$\langle$c3,Bob,Al,S_b(e),[price(e,low),warranty(e,low)]$\rangle$,$\emptyset$)	M_1
M_6	Al	Bob	accept(good_deal,$\langle$c3,Bob,Al,S_b(e),[price(e,low),warranty(e,low)]$\rangle$,$\emptyset$)	M_5

to this knowledge the A-type agent Al is able to consider which of these services could satisfy its goals.

The middle of Table 3 depicts the speech acts exchanged between Al and Alice playing an information-seeking dialogue. This dialogue corresponds to step 2 of the e-procurement process (cf Table 2). The first move is for Al to pose a question to Alice, M_0. This locution seeks the testimony of Alice about Bob. This testimony is high (cf M_1) and argued by the number of previous experiences and their values.

The bottom of Table 3 depicts the speech acts exchanged between Al and Bob playing a negotiation dialogue. This dialogue corresponds to the step 6 of the e-procurement process (cf Figure 2). They attempt to come to an agreement on the contract for the provision of a service S_b to reach the common goal good_deal. A contract is a tuple $\langle$cid, debtor, creditor, service, terms$\rangle$ where cid is the contract identifier, debtor is the agent providing the service, creditor is the agent requesting the service. A contract is concerned by the provision of a service provided that the list of terms & conditions (denoted terms) are satisfied. With the message M_1, Bob informs Al that it finds out that the terms & conditions of the contract for the provision of the service $S_b(d)$ are justified with respect to the common goal (good_deal). However, Al does not find S_b(d) justified and he proposes S_b(c). Since none of these proposals have been jointly accepted, they should not be considered in the following of the negotiation. Bob attempts to determine the reasons for Al's choice (cf M_3) which is the cost (rather than the quality of service). Given Al's response in M_4, Bob includes the goal provided by Al. Therefore, it finds between the other solutions (S_b(e) and S_b(f)) the one preferred by Al (S_b(e)) and suggest it (M_5). Finally, Al communicates his agreement with the help of an accept (M_6) which closes the dialogue.

5 Case Run

We consider here a case run which illustrates the agent deliberation and communication in the scenario of section 2. This case run involves `Al`, an A-type agent, and `Bob`, a B-type agent, negotiating the provision of a service S_b.

The terms & conditions considered for the evaluation of the contract about S_b during the negotiation (cf bottom of Table 3) are represented at the two axis of the two dimension plot in Figure 10. The acceptability space of the two participants is represented by shaded areas and depends on the price (y-axis) and the warranty (x-axis). Four points reflect the combinations of values: $S_b(\mathtt{c})$ where warranty is high and price is low, $S_b(\mathtt{d})$ where warranty is low and price is high, $S_b(\mathtt{e})$ where both warranty and price are low and $S_b(\mathtt{f})$ where both are high. After the message M_2 (cf left of Figure 3), `Bob` only finds $S_b(\mathtt{d})$ justified and `Al` only finds $S_b(\mathtt{c})$ justified. After the message M_3 (cf center of Figure 10), the acceptability spaces of the two agents have shifted since neither $S_b(\mathtt{c})$ nor $S_b(\mathtt{d})$ have been jointly accepted. Both of the agents make concession since the MARGO mechanism allow to relax the preferences. After the message M_6 (cf right of Figure 10), both agents has identified $S_b(\mathtt{e})$ as a common solution. We can notice that the influence of `Al` on `Bob` avoid to explore the alternative $S_b(\mathtt{f})$ which is not justified from `Al`'s viewpoint. The influence of `Al` on `Bob` is supported by the extra information carry out by the argument.

We have illustrated here the main strength of argumentation-based negotiation which is, as pointed out by [53], that it allows agents to influence each other. Moreover, our agents make concessions when necessary as suggested by [2].

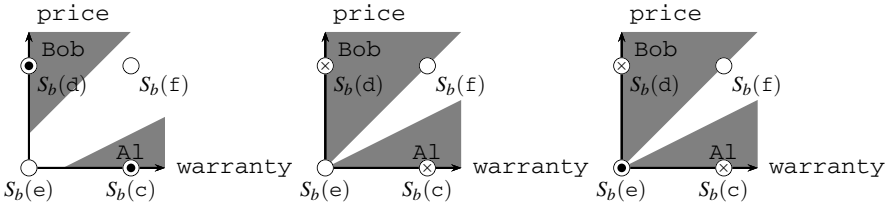

Fig. 10 Acceptability space of participants after the messages M_2 (left), M_3 (center) and M_6 (right)

Figure 11 represents the UML sequence diagram associated with the A-agent `Al`. For brevity, we only describe the internal mechanisms for the negotiations (step 6) of the e-procurement processes involving `Al` and `Bob`. The first transition shown is from the GUI to the IDMM. This is how the user *delegate*s the task of contracting the provision of a service to the agent. Using the techniques described in section 4.1, the user's competency is computed and sent to the SDMM. The SDMM bootstraps the SIM's protocol execution by invoking the protocol, agents it would like to communicate with and the role it will play within the protocol (i.e. the requester). The SIM then drives the interaction by sending the appropriate locution to the CM which

is relayed to the addressed B-type agent. During this exchange, decision must be made, the SIM delegates control to the SDMM to determine the correct course of action. This is shown in the figure 11 as "evaluate". Once the evaluation of the *contract* or *challenge* is performed by the SDMM, it provides the response which the SIM which encapsulates with the correct locution according to the interaction protocol. Finally, through the IDMM and the GUI, the user is notified of the provision of the service. The provider of the service, according to the semantics of the protocol, knows the contract and the provision of the service is accepted by the agent communicating locution, `accept`.

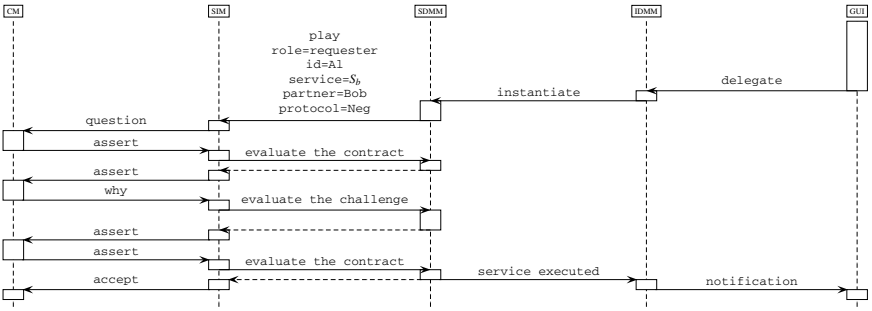

Fig. 11 Sequence diagram of `Al`

6 Deployment and Implementation

Agents in ArguGRID are deployed in GOLEM, a prototype agent platform which can support complex applications of cognitive agents interacting within a distributed environment [10]. GOLEM agents are represented by mean of an agent mind, programmed in Prolog, connected to a Java agent body. The agent body incorporates sensors, to perceive the events happening in the agent environment, and effectors, to produce events in the agent environment. The connection between the Java body and the agent mind is done by using a Prolog-Java bridge, such as InterProlog. In particular in this Chapter we made use of a SWI Prolog agent mind, loading both the MARGO and LCC Prolog libraries, connected to a GOLEM agent body by means of InterProlog. In this section, we also describe how GOLEM has been extended to take into account the specific context of the services.

GOLEM has been integrated with the P2P platform PLATON [36] allowing agents to discover service provider agents. These agents may be deployed in different *GOLEM containers* distributed over a network (cf Figure 12). In order to support agent/service discovery, each container includes two types of registries.

KDE semantic registry: a database which holds semantic descriptions of available services. Such a registry is in charge to perform semantic matchmaking between service queries and concrete available services.

GOLEM registry: a database working as a cache on top of the PLATON P2P platform. Its function is holding information about agents discovered in the GOLEM distributed environment.

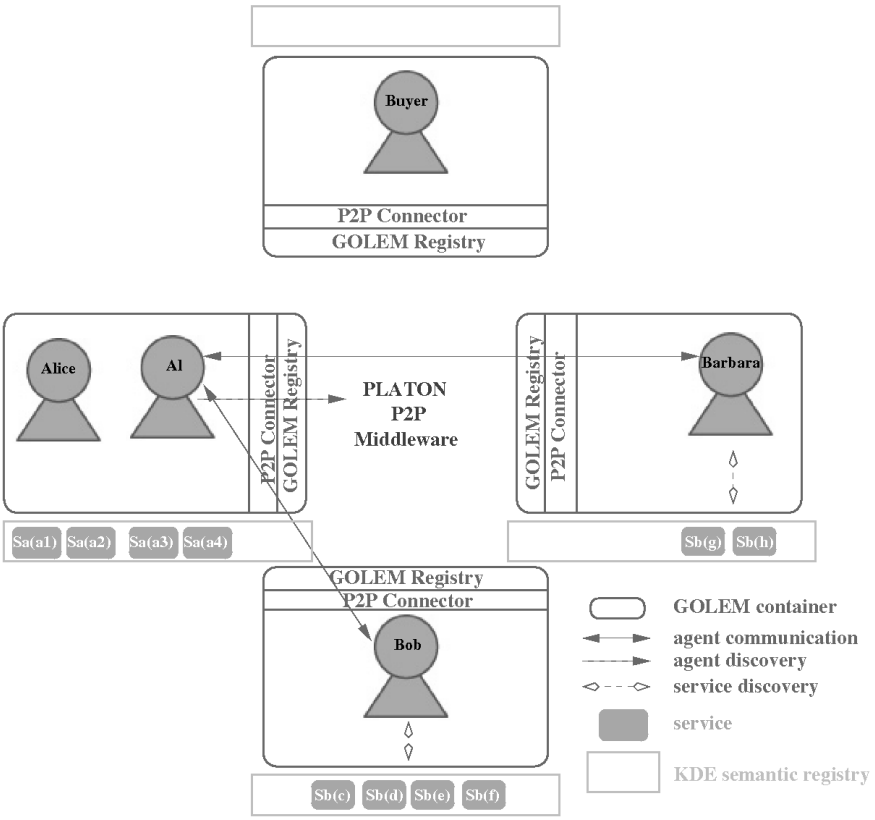

Fig. 12 System Deployment

The *KDE semantic registry* works as a catalogue for the provider agents, holding information about the services provided by the organisation (i.e. user) represented by the agent. In order to support the matching of service descriptions and concrete services, the automated search, the services inside the KDE semantic registry have to carry sufficient information that describes them. Within the agent framework, we abstract away from the particular formalism utilised in KDE to describe the Web services, performing a translation to Prolog terms at the interface between KDE and the GOLEM agent environment. As specified in [10, 11] GOLEM agents can reason about C-logic structures [12]. C-logic is a convenient specification language that has a direct translation to first-order logic and Prolog (see [10, 11] for more details about

the use of C-logic in GOLEM) and that is used in GOLEM to deal with complex structures represented as logical objects. In the context of ArguGRID, the following C-logic structure:

web_service:w1[
service_type ⇒ connection,
domain ⇒ eprocurement,
organisation ⇒ argugrid,
precondition ⇒ creditcard:CCard,
postcondition ⇒ connection:Con[price ⇒ P, warranty ⇒ W]
constraints ⇒ {W= 24*h*, P < 10€}]

represents an instance w1 of class web_service that expresses the requirements for a Web service of type connection. Such a Web service has to be in the eprocurement domain, provided by the argugrid organisation, taking in input objects of class creditcard and providing as output an object of class connection, with a price P and a warranty W that have associated a set of constraints for which the price has to be less than 10 euros per month and a warranty of 24 hours of assistance.

Agents deployed inside a GOLEM container have access to a *P2P connector* interface. This interface wraps PLATON inside GOLEM and allows agents to perform queries to discover other agents and services. The result of such a query is stored inside GOLEM registries for future use. Service provider agents are discovered according to their *affordances* [10], properties that describe the role and competencies of the agent. In our current deployment of ArguGRID we use three properties to describe agents:

- service types, the types of services an agent can represent (e.g. {internet_connection, computer_system});
- organisation, the organisation in which the agent belongs to (e.g. argugrid);
- domain, the domain of knowledge the agent is competent about (e.g. e-procurement).

The above three fields are specified inside the query proposed by the requester, under the parameters of the web_service instances, as previously presented. At the lower level these properties are used by PLATON to create a K-D tree index of the agents; this index ensures that the steps to access a point (an agent) is logarithmic to the number of peers (GOLEM containers) deployed in the distributed network. The details of the algorithm used to describe this process are beyond the scope of this work. The interested reader is referred to [35] for more details.

At the higher level an agent can connect to other agents belonging to different containers by performing physical actions on a connector object (see [10]). Conceptually, the effects of such an action is similar to that of a human manipulating a physical object in a real environment. However, here the effects of the action on the connector make present agents of other containers using PLATON to perform the discovery. When a requester looks for potential suppliers (cf step 1 of the procurement process described in Figure 2 of Section 2), the requester queries the registry

Table 4 Interaction with the Connector Interface

M_k	S_k	H_k	A_k	R_k
M_0	`Al`	`Registry`	`query(supplier,proposal(`x,y`),[`$S_b(y)$`])`	θ
M_1	`Registry`	`Al`	`answer(supplier,proposal(Barbara,`y`),[`$S_b(y)$`])`	M_0
M_2	`Registry`	`Al`	`answer(supplier,proposal(Bob,`y`),[`$S_b(y)$`])`	M_0

and the corresponding connector is involved. Table 4 shows the physical actions exchanged between `Al` and the registry. The first move is for `Al` to pose a query to the registry, M_0. This physical action seeks the suppliers for the available services S_b. The registry reacts to `Al`'s query with several physical actions with inside the description of the various suppliers (M_1 and M_2) in terms of their affordances. According to the new knowledge, the dialogical commitment store of `Al` is updated and this latter is able to perform the next step of the procurement process which consists of collecting information on the available suppliers in order to short-list them. Figure 12 exemplifies what stated above. The A-type agent `Al` queries the connector to find the B-type agents. As a result of these interactions `Al` is notified of the new agents discovered, `Bob` and `Barbara`. Once these agents are discovered, `Al` queries them in such a way that they can check their private KDE semantic registry with an interaction similar to the one presented above with the connector. After the KDE semantic registry provides the services that can match the requester requirements, the agents start the negotiation.

7 Related Work

Combining service-oriented computing and architectures with software agents is an active area of research for intelligent systems [15, 51]. More specifically, current visions of Web-services and the role of agents [14] predict important implications in the engineering of complex distributed systems [27] in general and Grid [22] and ubiquitous [28] computing in particular. A large part of this effort focuses on the service selection problem, where a computational logic approach is playing an important role, for example see McIlraith and colleagues [43, 42], Baldoni et al [5], and Lomuscio et al [34]. In contrast with these efforts, we advocate the automatic discovery of services by agents and how the selection of these services can be made concrete within the context of an argumentation architecture that agents can utilise given abstract specifications of users' goals. The advantage of using argumentation is that agents can provide supporting arguments to select a service, thus being in a position to provide reasons for why a particular service has to be selected instead of another.

The importance of service selection has been studied by Sreenath and Singh in [56]. They explain how services differ from products in terms of how they are being discovered, delivered and evaluated. Sreenath and Singh also provide a general framework for service selection that combines conventional approaches such as

reputation systems, collaborative filtering, and P2P systems with novel techniques from lattice theory.

Maximilien and Singh in [38, 39] argue that current techniques provide no support to actually make rational selections of services, which are key to accomplishing autonomic behaviour in service-oriented computing. They develop a multi-agent system framework based on an ontology for QoS and a new model of trust. The ontology provides a basis for providers to advertise the offered services, for consumers to express their preferences, and for ratings to be gathered and shared. Our work is complementary to the effort of Singh and colleagues in that we provide the logic-based reasoning and social capabilities of an agent to prioritise preferences and select the services that best match the goals of a consumer user. In this context, we also implement these ideas and we can deploy multiple agents that interact and communicate with one another using a platform that can discover these agents over a complex network.

Baldoni and colleagues [6] address the problem of automatic selection and composition of Web services, discussing the advantages that derive from the inclusion, in a Web service declarative description, of the high-level communication protocol, that is used by the service for interacting with its partners, allowing a rational inspection of it. The approach they propose is set in the context of semantic Web by capitalising on existing research in multi-agent systems. Similarly to our work, Web services are viewed as (represented by) software agents, communicating by predefined shareable interaction protocols. A logic programming framework based on modal logic is proposed, where the protocol-based interactions of Web services are formalised and the use of reasoning about actions and change techniques for performing the tasks of selection and composition of Web services in a way that is personalised to a user's request. Like our work, by applying reasoning techniques on a declarative specification of the service interactions allows to gain flexibility in fulfilling the user preference in the context of a Web service matchmaking process. However, this work focuses on discovery and service selection but not on service composition, that we plan to consider in future works.

Bentahar et al. [8] use argumentation implemented via software agents to reason about Web services and improve their performance through the notion of *communities*. A community of Web services is a set of services with similar functionality that grouped together to facilitate discovery. Each community is organised in terms of a *master* Web service (a bit like a broker of composite Web-services) that argues with a set of *slave* Web services (a bit like basic Web services that can participate in composite ones). To persuade a Web service to be part of a composite Web service in a community, master and slave Web services use persuasion and negotiation techniques associated with their argumentation abilities. Web services interact flexibly via dialogue games and are implemented as JACK agents [26]. We differ from this approach in that we use a calculus, LCC, to flexibly handle the interactions. We focus on the use of argumentation to build the decision capabilities of an agent that Bentahar et al abstract away from. We do not need communities to index Web service for discovery but we use GOLEM containers combined with a P2P

framework to enable service discovery. Finally, community for us is a social aspect of the system that emerges via communication, as in [37].

Other MAS research takes a strictly societal or communicative view of agency. It models and formalizes the communication and interaction between agents. The paper [41] describes a formal model for the formation of virtual organisations. The formal model, although accommodates the various technologies described in this chapter, it does not rely upon them. This work is in contrast to other approaches. For example, the authors discuss in [47] as part of the CONTRACT project a contract-based approach to enforcing normative behaviour within multi-agent systems. Numerous works that describe agent systems as electronic institutions [17].

The model and architecture of argumentative agents we have presented, contrary to the abstract negotiation framework of [2], have been implemented and tested with real-world use cases. Our proposal is not the first attempt in this direction. For instance, Kakas and Moraitis [30] provide a framework for effective argumentation-based negotiation. With respect to the latter, we have introduced multi-criteria techniques for the decision making related to the evaluation of proposals by the agents. The framework presented here can be seen to some extent as a specialisation of [2], tailored to service selection. For this purpose, we have proposed: i) an argumentation-based mechanism for multi-criteria decision-making integrating assumptions; ii) an general architecture instantiated with argumentation and logic techniques; iii) the deployment of our architecture with a MAS platform.

We have used here the argumentation-based mechanism for multi-criteria decision-making integrating assumptions proposed in [45]. Contrary to [1, 3, 4], our framework incorporates abduction on missing information, as suggested by [29]. This property is required by the IDMM to build arguments upon suppositions. We can deploy our framework for a number of argumentation semantics by relying on [20], whereas [29] is committed to one such semantics. [48] propose a critical survey of some computational models of argumentation over actions. For instance, [3, 4] have considered several principles according to the different types of arguments which are considered are aggregated. However, contrary to our approach, the potential interaction between arguments is not considered. We have considered the example borrowed from [58] and we have adopted, like [7], an abductive approach to practical reasoning/decision-making which is directly modelled within in our framework.

We have proposed a general architecture which distinguishes the internal reasoning, the social reasoning, and the communication. This architecture has been instantiated with argumentation and logic techniques. The KGP model [31, 33] adopts Knowledge, Goals and Plans as the main components of an agent state. There is no gap between the logical specification of KGP agents and their implementations. Indeed, this model uses computational logic frameworks extending logic programming for specification and realisation purposes. However, this model deals only partially with priorities as required by service selection applications, namely only with preferences between goals [32], but not with uncertainty of knowledge and expected

utilities of alternative services. MARGO[13], i.e. the implementation of our argumentation framework, provided here a revised representation of knowledge, goals and decisions without planning abilities as required by our application. MARGO uses the implementation of [20] in the CaSAPI system [23]. We use LCC for representing and enacting protocols to allow the social norms used by the agents to be verifiable, communicable, inspectable and potentially modifiable. The merits of using a first class protocol are described in [44]. An extension of KGP for service composition has been discussed in [60].

As in [54], we have presented the deployment of an architecture with a MAS platform. The Web-service environment in GOLEM[14] builds upon previous work on deploying MAS with PROSOCS [9]. However, GOLEM extends PROSOCS in many ways. Apart from rationalising the PROSOCS mind/body architecture [57], our implementation uses an argumentative mind component rather than the KGP implementation discussed in [9]. It also generalises the interaction of agents with objects and uses containers as locations [49] of resources, including services. In this sense, GOLEM is part of a growing research and development effort to model situated multi-agent systems [63] without abstracting away the notion of the agent environment as some popular platforms do (e.g. JADE [21] or RETSINA [59]).

8 Conclusion and Future Work

In this chapter, we have described an agent model and architecture using argumentation to automate the selection of services and partners. The modular design of the deliberative and communicative processes bring the well known engineering benefits of modularity as well as create a unique model of agents. The three internal modules are dedicated respectively to decision making (for the IDMM), negotiation (for the SDMM), and communication (for the SIM), and are all realised using argumentation-based frameworks. If the IDMM can be considered as the base appetite of the agent reasoning about how to achieve its individualistic goals and the SDMM can be considered as the social reasoner, conscious of goal solving through collaboration, then the SIM can be considered as its social conscience and filters those impulses by following the social norms of the agent society. For our purposes, the social norms are the rules of dialogues encoded as interaction protocols. In order to test this approach, the architecture is instantiated in the ArguGRID project, by means of the argumentative decision making tool MARGO (for the first two modules) and by the LCC tool for enforcing protocol conformance (for the last module).

We use GOLEM [10] for the deployment of our agents. GOLEM is a multi-agent platform which can support complex application through the deployment of cognitive agents situated in a distributed environment over a network. In particular, GOLEM allows a declarative description of resources in the agent environment, so that such a description is understandable by cognitive agents programmed following the patterns of logic programming. In this way, agents can discover each other

[13] http://margo.sourceforge.net
[14] http://golem.cs.rhul.ac.uk

in a distributed network, as well as reasoning about complex structures representing the requirement of the users in terms of Web services. One possible future work direction, is to include reasoning about semantic descriptions of Web services using OWL [16] in our ArguGRID agents, following an approach like the one described in [25]. The main advantages of having agents reasoning about semantic descriptions, would be that we can decouple the description of the Web services from their implementation and that our cognitive model could be reused in different platforms than ArguGRID.

As mentioned previously, contracts create virtual organisations consisting of debtors and creditors. In this chapter, we have focused on the agent model and architecture and we have ignored virtual organisations. A formal model for virtual organisations using agent negotiation to determine its configuration is described in [41]. The decision making abilities of our agents is useful during the operation of virtual organisations for exception handling, which is labelled as reformation. The additional tasks of monitoring and reportage during the operation and dissolution of the virtual organisations produce data about its performance and form the basis for its evaluation. The execution task during the operation phase is the coordination of the delivery of services.

We have learned a number of lessons while developing the ArguGRID prototype system. Firstly, the knowledge engineering process can become complex when dealing with multiple and possibly heterogeneous knowledge representation technologies. In ArguGRID we decided to avoid formalisms based on Description Logic, for instance, as the reasoning that we could perform using this type of technology was limited to subsumption mechanisms; MARGO style argumentation combined with the underlying Prolog engine allowed for a computationally more expressive and flexible approach. Secondly, another problem that we discovered is that P2P engines have several limitations due to the impossibility to handle semantic descriptions of the entities, but only flat multidimensional points. Thirdly, if distribution is an important issue and/or access to the communication protocols is predicated on an agent's role then a system like LCC is more appropriate for dealing with agent communication. Alternatively, if agents are permitted to access the whole protocol then a shared memory approach [61] is equally suitable, can limit the amount of messages exchanged and can mediate the protocol of communication. Finally, the definition of a mediation infrastructure, such as the one provided by GOLEM, allows for a better integration of the components involved in the system, as it allows for a better description and representation of the resources.

Acknowledgements. This work was supported by the Sixth Framework IST programme of the EC, under the 035200 ArguGRID project.

References

1. Amgoud, L.: A general argumentation framework for inference and decision making. In: Fahiem Bacchus, T.J. (ed.) Proc. of the 21st Conference on Uncertainty in Artificial Intelligence (UAI), pp. 26–33. AUAI Press, Edinburgh (2005)

2. Amgoud, L., Dimopoulos, Y., Moraitis, P.: A unified and general framework for argumentation-based negotiation. In: Proc. 6th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Honolulu, Hawaii, pp. 963–970 (2007)
3. Amgoud, L., Prade, H.: Comparing decisions in an argumentation-based setting. In: Proc. of the 11th International Workshop on Non-Monotonic Reasoning (NMR), Session on Argumentation, Dialogue, and Decision Making, Lake District, UK, pp. 426–432 (2006)
4. Amgoud, L., Prade, H.: Explaining qualitative decision under uncertainty by argumentation. In: Proc. of the 21st National Conference on Artificial Intelligence (AAAI), Boston, pp. 16–20 (2006)
5. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: Reasoning about interaction protocols for web service composition. Electr. Notes Theor. Comput. Sci. 105, 21–36 (2004)
6. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: Reasoning about interaction protocols for customizing web service selection and composition. J. Log. Algebr. Program. 70(1), 53–73 (2007)
7. Bench-Capon, T., Prakken, H.: Justifying actions by accruing arguments. In: Proc. of the 1st International Conference on Computational Models of Argument, pp. 247–258. IOS Press (2006)
8. Bentahar, J., Maamar, Z., Benslimane, D., Thiran, P.: An argumentation framework for communities of web services. IEEE Intelligent Systems 22(6), 75–83 (2007)
9. Bracciali, A., Demetriou, N., Endriss, U., Kakas, A., Lu, W., Stathis, K.: Crafting the mind of PROSOCS agents. Applied Artificial Intelligence 20(2-4), 105–131 (2006)
10. Bromuri, S., Stathis, K.: Situating cognitive agents in GOLEM. In: Weyns, D., Brueckner, S., Demazeau, Y. (eds.) Proc. of the Engineering Environment-Mediated Multiagent Systems Conference (EEMMAS), pp. 76–93. Katholieke Universiteit Leuven, Leuven (2007)
11. Bromuri, S., Stathis, K.: Distributed Agent Environments in the Ambient Event Calculus. In: DEBS 2009: Proceedings of the Third International Conference on Distributed Event-Based Systems. ACM, New York (2009)
12. Chen, W., Warren, D.S.: C-logic of Complex Objects. In: PODS 1989: Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 369–378. ACM Press, New York (1989)
13. Clemen, R.T.: Making Hard Decisions. Duxbury Press (1996)
14. Cohen, M., Stathis, K.: Strategic change stemming from e-commerce: Implications of multi-agent systems in the supply chain. Strategic Change 10, 139–149 (2001)
15. Curcin, V., Ghanem, M., Guo, Y., Stathis, K., Toni, F.: Building next generation Service-Oriented Architectures using Argumentation Agents. In: 3rd International Conference on Grid Services Engineering and Management (GSEM 2006), Germany (2006)
16. Dean, M., Schreiber, G., Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL web ontology language reference. Tech. rep., W3C (2004), `http://www.w3.org/TR/2004/REC-owl-ref-20040210/`
17. Sierra, C., Dignum, F. (eds.): AgentLink 2000. LNCS (LNAI), vol. 1991. Springer, Heidelberg (2001)
18. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. 77(2), 321–357 (1995)
19. Dung, P.M., Kowalski, R.A., Toni, F.: Dialectic proof procedures for assumption-based, admissible argumentation. Artificial Intelligence 170(2), 114–159 (2006)
20. Dung, P.M., Mancarella, P., Toni, F.: Computing ideal sceptical argumentation. Artificial Intelligence, Special Issue on Argumentation 171(10-15), 642–674 (2007)

21. Bellifemine, F.L., Giovanni Caire, Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley (2007)
22. Foster, I.T., Jennings, N.R., Kesselman, C.: Brain meets brawn: Why grid and agents need each other. In: 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), pp. 8–15. IEEE Computer Society, New York (2004)
23. Gartner, D., Toni, F.: CaSAPI: a system for credulous and sceptical argumentation. In: Simari, G., Torroni, P. (eds.) Proc. of the Workshop on Argumentation for Nonmonotonic Reasoning (ArgNMR), pp. 80–95 (2007)
24. Ghanem, M., Azam, N., Boniface, M., Ferris, J.: Grid-enabled workflows for industrial product design. In: Proc. of the 2nd IEEE International Conference on e-Science and Grid Computing (e-Science 2006). IEEE Computer Society (2006)
25. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: Proc. of the Twelfth International World Wide Web Conference (WWW), pp. 48–57. ACM (2003)
26. Howden, N., Ronnquist, R., Hodgson, A., Lucas, A.: Jack - summary of an agent infrastructure. In: 5th International Conference on Autonomous Agents (2001)
27. Huhns, M.N., Singh, M.P.: Service-oriented computing: Key concepts and principles. IEEE Internet Computing 9(1), 75–81 (2005)
28. Huhns, M.N., Singh, M.P., Burstein, M.H., Decker, K.S., Durfee, E.H., Finin, T.W., Gasser, L., Goradia, H.J., Jennings, N.R., Lakkaraju, K., Nakashima, H., Parunak, H.V.D., Rosenschein, J.S., Ruvinsky, A., Sukthankar, G., Swarup, S., Sycara, K.P., Tambe, M., Wagner, T., Gutierrez, R.L.Z.: Research directions for service-oriented multiagent systems. IEEE Internet Computing 9(6), 65–70 (2005)
29. Kakas, A., Moraitis, P.: Argumentative-based decision-making for autonomous agents. In: Proc. of the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 883–890. ACM Press (2003)
30. Kakas, A., Moraitis, P.: Adaptive agent negotiation via argumentation. In: Proc. 5th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Hakodate, Japan, pp. 384–391 (2006)
31. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: The KGP model of agency. In: Proc. of ECAI, pp. 33–37 (2004)
32. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: Declarative Agent Control. In: Leite, J., Torroni, P. (eds.) CLIMA 2004. LNCS (LNAI), vol. 3487, pp. 96–110. Springer, Heidelberg (2005)
33. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: Computational logic foundations of kgp agents. J. Artif. Intell. Res. (JAIR) 33, 285–348 (2008)
34. Lomuscio, A., Qu, H., Sergot, M.J., Solanki, M.: Verifying Temporal and Epistemic Properties of Web Service Compositions. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 456–461. Springer, Heidelberg (2007)
35. Lymberopoulos, L., Bromuri, S., Stathis, K., Kafetzoglou, S., Grammatikou, M.: Towards a p2p discovery framework for an argumentative agent technology assisted grid. In: Proc. of the CoreGRID Workshop on Grid Programming Model, Grid and P2P systems Arhcitectures, Grid Systems, Tools, and Environments, Crete, Greece (2007)
36. Lymberopoulos, L., Papavassiliou, S., Maglaris, V.: A novel load balancing mechanism for P2P networking. In: Proc. of ACM Sponsored Conference GridNets, Lyon, France (2007)
37. Mamdani, E., Pitt, J., Stathis, K.: Connected Communities from the standpoint of Multiagent Systems. New Generation Computing 17(4) (1999)
38. Maximilien, E.M., Singh, M.P.: A framework and ontology for dynamic web services selection. IEEE Internet Computing 8(5), 84–93 (2004)

39. Maximilien, E.M., Singh, M.P.: Toward autonomic web services trust and selection. In: Aiello, M., Aoyama, M., Curbera, F., Papazoglou, M.P. (eds.) Service-Oriented Computing - ICSOC 2004, pp. 212–221. ACM, New York (2004)
40. McBurmey, P., Parsons, S.: Games that agents play: A formal framework for dialogues between autonomous agents. Journal of Logic, Language and Information 11(3), 315–334 (2002), Special Issue on Logic and Games
41. McGinnis, J., Stathis, K., Toni, F.: A formal model of agent-oriented virtual organisations and their formation. Multiagent and Grid Systems 7(6), 291–310 (2011)
42. McIlraith, S.A., Son, T.C.: Adapting golog for composition of semantic web services. In: Fensel, D., Giunchiglia, F., McGuinness, D.L., Williams, M.A. (eds.) Proceedings of the Eighth International Conference on Principles and Knowledge Representation and Reasoning (KR 2002), pp. 482–496. Morgan Kaufmann, France (2002)
43. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic web services. IEEE Intelligent Systems 16(2), 46–53 (2001)
44. Miller, T., McBurney, P.: Using Constraints and Process Algebra for Specification of First-Class Agent Interaction Protocols. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS (LNAI), vol. 4457, pp. 245–264. Springer, Heidelberg (2007)
45. Morge, M.: The Hedgehog and the Fox. In: Rahwan, I., Parsons, S., Reed, C. (eds.) ArgMAS 2007. LNCS (LNAI), vol. 4946, pp. 114–131. Springer, Heidelberg (2008)
46. Morge, M.: Arguing over goals for negotiation: Adopting an assumption-based argumentation decision support system. In: Jao, C. (ed.) Efficient Decision Support Systems - Practice and Challenges From Current to Future, ch. 12, pp. 211–240. InTech (2011)
47. Oren, N., Panagiotidi, S., Vázquez-Salceda, J., Modgil, S., Luck, M., Miles, S.: Towards a formalisation of electronic contracting environments. In: COIN@AAMAS&AAAI, pp. 156–171 (2008)
48. Ouerdane, W., Maudet, N., Tsoukias, A.: Arguing over Actions That Involve Multiple Criteria: A Critical Review. In: Mellouli, K. (ed.) ECSQARU 2007. LNCS (LNAI), vol. 4724, pp. 308–319. Springer, Heidelberg (2007)
49. Overeinder, B.J., Brazier, F.M.T., Marin, O.: Fault tolerance in scalable agent support systems: Integrating darx in the agentscape framework. In: Proc. of the 3rd IEEE International Symposium on Cluster Computing and the Grid (CCGrid), pp. 688–696. IEEE Computer Society, Japan (2003)
50. Papazoglou, M.P.: Service-oriented computing: Concepts, characteristics and directions. In: 4th International Conference on Web Information Systems Engineering (WISE 2003), pp. 3–12. IEEE Computer Society, Italy (2003)
51. Payne, T.R.: Web services from an agent perspective. IEEE Intelligent Systems 23(2), 12–14 (2008)
52. Rahwan, I.: Argumentation in multi-agent systems. Guest Editorial: Autonomous Agents and Multiagent Systems 11(2), 115–125 (2005)
53. Rahwan, I., Ramchurn, S.D., Jennings, N.R., McBurney, P., Parsons, S., Sonenberg, L.: Argumentation-based negotiation. The Knowledge Engineering Review 18(4), 343–375 (2003)
54. Ricci, A., Buda, C., Zaghini, N., Natali, A., Viroli, M., Omicini, A.: simpa-ws: An agent-oriented computing technology for ws-based soa applications. In: Paoli, F.D., Stefano, A.D., Omicini, A., Santoro, C. (eds.) Proceedings of the 7th WOA 2006 Workshop, From Objects to Agents (Dagli Oggetti Agli Agenti), *CEUR Workshop Proceedings*, CEUR-WS.org, Italy (2006)

55. Robertson, D.: Multi-agent Coordination as Distributed Logic Programming. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3132, pp. 416–430. Springer, Heidelberg (2004)
56. Sreenath, R.M., Singh, M.P.: Agent-based service selection. Journal of Web Semantics 1(3), 261–279 (2004)
57. Stathis, K., Kakas, A.C., Lu, W., Demetriou, N., Endriss, U., Bracciali, A.: PROSOCS: a platform for programming software agents in computational logic. In: Müller, J., Petta, P. (eds.) Proceedings of the Fourth International Symposium From Agent Theory to Agent Implementation (AT2AI 2004 – EMCSR 2004 Session M), Vienna, Austria, pp. 523–528 (2004)
58. Stournaras, T. (ed.): eBusiness application scenarios. Deliverable document D1.2 ARGUGRID (2007)
59. Sycara, K., Paolucci, M., Velsen, M.V., Giampapa, J.: The retsina mas infrastructure. Autonomous Agents and Multi-Agent Systems 7(1-2), 29–48 (2003)
60. Toni, F.: Argumentative kgp agents for service composition. In: Proc. AITA 2008, Architectures for Intelligent Theory-Based Agents, AAAI Spring Symposium. Stanford University, USA (2008)
61. Urovi, V., Stathis, K.: Playing with agent coordination patterns in MAGE. In: Coordination, Organization, Institutions and Norms in Agent Systems (COIN@AAMAS 2009), Budapest, Hungary (2009)
62. Vreeswijk, G.: Abstract argumentation systems. Artificial Intelligence 90(1-2), 225–279 (1997)
63. Weyns, D., Omicini, A., Odell, J.: Environment as a first-class abstraction in multi-agent systems. Autonomous Agents and Multi-Agent Systems 14(1), 5–30 (2007)