INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS
VOL.12, NO.2 JUNE 2007 130-141

The ORCAS e-Institution: a Platform to Develop Open, Reusable and Configurable Multi-Agent Systems

Mario GOMEZ and Enric PLAZA

Abstract—**The concepts and methodology needed for designing, developing, and implementing real life applications based on multi-agent systems are today still a challenge for researchers in Artificial Intelligence and Computer Science. Industrial-strength multi-agent systems require, among other things, reusability, i.e. the capability of not having to design and implement from scratch a new multi-agent system for every new application domain. In order to improve the reusability of multi-agent systems, we have taken a knowledge modelling stance coming from the research on knowledge engineering, and adapted its insights to deal with multi-agent systems. The resulting contributions are organized in the Open, Reusable and Configurable Agent Systems (ORCAS) framework, comprising conceptual guidelines to design, methodological commitments for development, and an implemented platform to make concrete these abstract notions and effectively experiment with them. This paper describes the ORCAS e-Institution, and agent platform for developing and deploying open, reusable and configurable cooperative multi-agent systems.**

Index Terms—**multi agent systems, knowledge engineering, agent platform, agent-oriented software engineering**

I. INTRODUCTION

Multi-Agent Systems (MAS) are habitually referred to as open systems, but most of the initial research has focused on closed systems [1], typically designed by one team for one homogeneous environment, with participating agents sharing common high-level goals in a single domain. The communication languages and interaction protocols are typically in-house protocols defined by the design team prior to any agent interactions; systems are scalable under controlled conditions and design approaches tend to be ad-hoc, rather than using any specific methodology. It is often suggested the need for real open systems that were capable of dynamically adapting themselves to changing environments. In a really open MAS the participants (both human and software agents) are unknown beforehand, may change over time and may be developed by different parties. However, according to the predictions of the European Network of Excellence for Agent Based Computing, real open MAS spanning multiple application domains and involving heterogeneous agents will not be achieved in a foreseeable future, and not before year 2009 [2]. This cautious prediction obeys to challenges yet to be (or very recently) undertaken, including: effective agreed standards to allow open agent systems, semantic infrastructure for open agent communities, reasoning capabilities for agents in open environments, agent ability to understand user requirements, agent ability to adapt to changes in the environment, and mechanisms to ensure trust. In order to overcome the limitations of current agent infrastructures and achieve the former goals, researchers must tackle several problems, including the following:

1) the *connection* (discovery and location) problem, or how to put providers and requesters in contact;
2) the *interoperability* problem, or how to achieve a meaningful interaction among heterogeneous agents at the syntactic, semantic and pragmatic levels;
3) the *coalition* problem, or how to form and coordinate agent teams to solve problems in a cooperative way;
4) the *reuse* problem, or how to use the same agent capabilities across several application domains;
5) the *accountability* problem, or how to predict/explain MAS based on problem requirements.

In order to deal with some of the problems above (mainly 1 and 2), the agent community is using an integrating architectural pattern developed by the software engineering community called *middleware*: connectivity software that allow multiple processes running on one or more machines to interact across a network. *Intelligent middleware* aims to achieve the highest degree of interoperability, when systems can identify and react to the semantics of data. In MAS the middleware layer is usually provided by *middle agents* [3] that mediate between requesters and providers of capabilities (e.g. brokers and matchmakers). In addition to middle agents, the notion of an *Agent Capability Description Language* (ACDL) has been introduced recently [4] as a key element to enable MAS interoperation in open environments. An ACDL is a shared language that allows heterogeneous agents to coordinate effectively across distributed networks. In the literature, an ACDL is defined as a language to describe both agent advertisements and requests, and is primarily used by middle agents to pair service-requests with service-providing agents.

In order to confront the problems enumerated above, we have developed a multi-layered framework for developing and deploying cooperative MAS called ORCAS, which stands for Open, Reusable and Configurable multi-Agent Systems [5]. The ORCAS framework explores the use of a Knowledge Modelling Framework (KMF) as the basis of an ACDL for describing and composing agent capabilities with the aim of

M. Gomez is with the Department of Computing Science, University of Aberdeen, King's College, Aberdeen AB24 3UE, UK. E-mail: mgomez@csd.abdn.ac.uk.

E. Plaza is with the Artificial Intelligence Research Institute, Spanish National Research Council, Campus UAB, 08193 Bellaterra, Spain. E-mail: enric@iiia.csic.es.

maximizing capability reuse (problem 4). Moreover, ORCAS supports the automatic, on-demand configuration of agent teams according to stated problem requirements (problem 5). The ORCAS KMF provides a semantic infrastructure which is specially well suited to deal with the connection, interoperability and reuse problems. In addition, ORCAS provides an Operational Framework (OF) that links concepts from KMF, Multi-Agent Systems and Cooperative Problem Solving. Specifically, given a problem to be solved, the ORCAS OF describes how a software designed to solve a given problem can be operationalized on demand by forming and instructing a team of agents to solve that problem in a cooperative way (problem 3).

As part of the ORCAS framework, we have implemented and agent platform called the ORCAS e-Institution. This platform is based on the *electronic institutions* formalism [6], which, intuitively, refers to a sort of virtual place where agents interact according to explicit conventions. An e-Institution is responsible for defining the rules of the game, to enforce them and impose the penalties in case of violation; it provides the social mediation layer required to achieve a successful interaction: interaction protocols, shared ontologies, communication languages and social behavior rules. The ORCAS e-Institution brings an added value to both requesters and providers of capabilities: on the one hand, requesters are freed of finding adequate providers and brings a single interface to the multiple and heterogenous providers; on the other hand, the institution provides an advertisement service to capability providers, provides a mediation service for the team formation process, and facilitates coordination during the teamwork activity, allowing agents to solve complex problems that cannot be achieved by any single agent alone.

Finally, we have implemented a case-study application to demonstrate the feasibility of the ORCAS framework in practice: the Web Information Mediator (WIM), a MAS-based configurable application to search bibliographic information on the Internet [7]. WIM is the result of linking a library of information search and aggregation capabilities provided by agents, with a specific application domain, such as medicine. In particular, in our experiments to validate the domain independence feature, we have used domain knowledge from medicine, and more specifically, Evidence-Based Medicine (EBM), and afterwards we have connected the same search capabilities to the Computer Science domain.

The Knowledge Modelling Framework and the Operational Framework has been described elsewhere ([8] and [9] respectively), so herein we focus on the implemented agent platform (the ORCAS e-Institution). Section §II reviews recent work on open agent infrastructures and capability description languages, §III provides an overview of the ORCAS framework, §IV describes the ORCAS e-Institution, and §V discuss the contributions of the proposed infrastructure.

II. Related work

Nowadays the Web is shifting the nature of software development to a distributed *plug-and-play* process. This change advocates the use of specific integration architectural patterns based on *middleware* software; enabling services that allow multiple processes running on one or more machines to interact across a network. Intelligent middleware aims at achieving the highest degree of interoperability, when systems can identify and react to the semantics of data; for this reason, many research communities are interested on semantic interoperability, including Multi-Agent Systems (MAS), Semantic Web Services, Cooperative Information Systems, and Grid Computing. In MAS, the middleware layer is provided by *middle agents* [3], a specialized kind of agents mediating between requesters and providers of services, such as matchmakers [10], facilitators[11], [12], and brokers[13]). Several infrastructures have been proposed based on middle agents, for instance:

- *UMDL* (University of Michigan Digital Library) uses a distributed architecture [14] for a digital library that can continually reconfigure itself as users, contents, and services come and go. This has been achieved by the development of a multi-agent infrastructure with agents that buy and sell services from each other by using commerce and communication services [15]. This framework, called the Service Market Society (SMS), allows for the decentralized configuration of an extensible set of users and services [16]. There are many types of agents in the UMDL agent architecture: (a) information agents specialized in complementary knowledge areas, (b) interface agents that support the user in specifying queries, and (c) task planning agents [17] that are able to perform matchmaking between queries and agent services. Services are described in Loom.
- *OAA* (Open Agent Architecture) is a framework for building flexible, dynamic communities of distributed software agents [18]. OAA enables a cooperative computing style wherein members of an agent community work together to perform computation, retrieve information, and serve user interaction tasks. Communication and cooperation between agents are brokered by one or more facilitators, which are responsible of matching requests from users and agents, against descriptions of the capabilities provided by other agents.
- *RETSINA* (Reusable Environment for Task-Structured Intelligent Networked Agents) is an open multi-agent architecture that supports communities of heterogeneous agents [19]. The RETSINA system has been implemented on the premise that agents should form a community of peers that engage in peer to peer interactions. Any coordination structure in the community of agents should emerge from the relations between agents, rather than as a result of the imposed constraints of the infrastructure itself. In accordance with this premise, RETSINA does not employ centralized control within the MAS; instead, RETSINA implements distributed infrastructure services that facilitate the interactions between agents, as opposed to managing them.
- *DECAF* (Distributed, Environment-Centered Agent Framework) is a toolkit which allows a principled software engineering approach to building Multi-Agent Systems. The toolkit provides a platform to design,

develop, and execute agents. DECAF provides the necessary architectural services of a large-grained intelligent agent: communication, planning, scheduling, execution monitoring, coordination, and eventually learning and self-diagnosis. This is essentially, the internal "operating system" of a software agent, to which application programmers have strictly limited access. The control or programming of DECAF agents is provided via a GUI called the Plan-Editor. In the Plan-Editor, executable actions are treated as basic building blocks which can be chained together to achieve more complex goals, in the style of a *Hierarchical Task Network*.

Typically, the infrastructures supporting open MAS include middle agents or some sort of middleware services to match requesters and providers of capabilities. The process of verifying whether the specification of a capability matches the specification of a goal or task to be achieved, is called matchmaking. The language used to specify both the requests (tasks or goals to achieve) and the advertisements (services or capabilities provided by agents) is called Agent Capability Description Language (ACDL)[4], a shared language allowing heterogeneous agents to effectively communicate across distributed networks. Some examples of ACDL follow:

- LDL++ is a logical deduction language similar to Prolog that is used by brokers in the Infosleuth [13] distributed agent architecture. LDL++ supports inferences about whether an expression of requirements matches a set of advertised capabilities.
- ICL, an extension of PROLOG, is the interface, communication, and task coordination language shared by OAA agents[20], [18]. OAA agents employ ICL to perform queries, execute actions, exchange information, and manipulate data in the agent community. Every agent participating in an OAA-based system defines and publishes its capabilities expressed in ICL, and these declarations are used by a facilitator to communicate with the agent and also for delegating service requests to the agent.
- LARKS [21] (Language for Advertisement and Request for Knowledge Sharing) is a language used by RETSINA [19] agents to pair service-requesting agents with service-providing agents that meet the requesting agents [4]. LARKS is capable of supporting inferences, and incorporates application domain knowledge in agent advertisements and requests.

Semantic matchmaking, which is based on the use of shared ontologies to annotate agent capabilities [22], improves the matchmaking process and facilitates interoperation. The first implemented ACDLs (LDL, ICL) did not include semantic information, so the matchmaking process was performed at the lexical or syntactical levels. Only recently have some ACDLs include semantic information (e.g. LARKS) to improve interoperability. However, the reuse of existing capabilities over new application domains is still difficult in these infrastructures because capabilities are associated to a specific application domain.

Our approach to matchmaking is driven by the idea of reuse: how to reuse a capability for different tasks, across several application domains, and in coordination with the capabilities provided by other agents. As a consequence we are interested in software engineering approaches that aim at maximizing reuse, such as the Component-Based Software Development (CBSD) paradigm. By enhancing the flexibility and maintainability of systems, the ultimate goal of CBSD is to reduce software development costs, assemble systems rapidly, and reduce the maintenance burden associated with the support and upgrade of large systems [23]. In the CBSD paradigm constructing an application involves the use of prefabricated pieces, perhaps developed at different times, by different people and possibly with different purposes. There is, however, a major problem with software composition, the so called *Bottom Up Design Problem* [24], defined as: given a set of requirements, find a set of components within a software library whose combined behavior satisfies the requirements. The fundamental difficulty when addressing this problem is how to decompose the requirements in such a way as to yield component specifications. A reverse approach is to search the space of all possible component compositions until one satisfying the requirements is found [25], [26].

The reuse problem has also been investigated from a Knowledge Modelling perspective; in particular, the Task-Domain-Method paradigm, which prevails in Knowledge Modelling Frameworks (KMF) [27], [28], [29], [30], advocates a compositional approach to the software development cycle.

III. Overview of the ORCAS Framework

ORCAS is a framework for open Multi-Agent Systems that maximizes the reuse of agent capabilities across multiple application domains, and supports the automatic, on-demand configuration of agent teams according to problem requirements. ORCAS is designed to support agent developers through all the stages of the development process, from the analysis to the implementation and deployment. ORCAS is presented as a multi-layered framework so as to bring flexibility to developers and system engineers, since they can use only some layers of the framework, and adapt other layers according to its own preferences and needs. Specifically, the ORCAS framework comprises three layers, namely: Knowledge Modelling Framework, Operational Framework and Institutional Framework:

1) The *Knowledge Modelling Framework* (KMF) proposes a conceptual and architectural description of multiagent systems at the knowledge level [31], abstracting the specification of components from their implementation details. In addition, the KMF introduces a bottom-up design process to configure a system out of elementary components, until a system configuration is found that satisfies the requirements of each specific problem being solved. This configuration process is used to design the structure of a team and the competence required of every team role.
2) The *Operational Framework* (OF) deals with the link between the specification of components in the KMF, and the operational aspects of Multi-Agent Systems. The OF extends the KMF to become a full-fledged

Agent Capability Description Language. In addition, the OF proposes a new model of the Cooperative Problem Solving process that includes a Team Design stage prior to the Team Formation stage that is usual in other frameworks.

3) The *Institutional Framework* provides an implemented infrastructure for developing and deploying cooperative Multi-Agent Systems according to the other layers (KMF and OF).

A. Knowledge Modelling Framework

The ORCAS Knowledge Modelling Framework (KMF) proposes a conceptual description of Multi-Agent Systems at the *knowledge level* [31], abstracting the specification of components from implementation details. The purpose of the *Knowledge Modelling Framework* (KMF) is twofold: on the one hand, the KMF is a conceptual tool to guide developers in the analysis and design of Multi-Agent Systems in a way that maximizes capability reuse; on the other hand, the KMF provides the basis for an Agent Capability Description Language (ACDL) supporting the automatic, on-demand configuration of agent teams according to stated problem requirements (named Team Design).

The ORCAS KMF is based on the *Task-Method-Domain* paradigm, which distinguishes among three classes of components: *tasks*, *problem-solving methods* (PSM) and *domain models*. In ORCAS, there are tasks and domain models, while PSMs are replaced by agent capabilities. Moreover, the KMF specifies the *matching relations* that can be established among components to build applications. Figure 1 depicts the architectural pattern established by the ORCAS KMF, showing both the components and the matching relations.

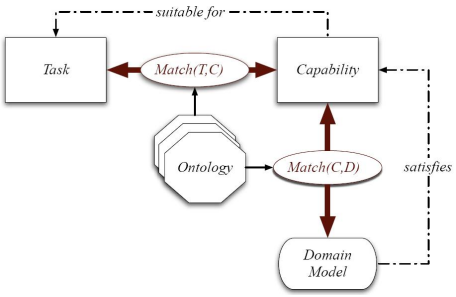


Fig. 1. The ORCAS Abstract Architecture

A *task* is a functional description of a class of problems to be solved. A task is functionally characterized by *input roles*, *output roles*, and the relationship between them, which is specified as a set of *preconditions* and *postconditions*.

A *capability* describes a particular method for solving certain class of problems. A capability is specified functionally by *input roles*, *output roles*, *preconditions* and *postconditions*. In addition, a capability can specify the type of domain knowledge (*knowledge-roles*) as well as some properties that have to be met by the domain knowledge to sensibly apply that capability (*assumptions*). There are two types of capability: *task-decomposer* and *skill*. A skill describes a primitive, atomic reasoning step, while a task-decomposer describes a decomposition of a class of problems (a task) into several smaller subproblems (subtasks).

Finally, a *domain model* (DM) describes the concepts, relations and properties characterizing the knowledge from a given application domain. A domain model is specified by a set of *knowledge-roles*, and some *properties* attributed to the domain knowledge (either verified or assumed to be true).

Task	T = $\langle ont, in, out, pre, post \rangle$
Capability	$C = \langle ont, in, out, pre, post, asm, kr \rangle$
Task-decomposer	$D = \langle ont, in, out, pre, post, asm, kr, st \rangle$
Skill	$S = \langle ont, in, out, pre, post, asm, kr \rangle$
Domain Model	$M = \langle ont, kr, prop \rangle$

Above we show the most important features used to specify components at the knowledge level, where ont is a set of ontologies defining the concepts that specify *signature elements* and *formulae* using certain *Object Language* $\mathbb{O}$ (e.g. First Order Logic), st is an ordered set of subtasks; in, out, kr are inputs, outputs and knowledge-roles respectively, specified as signature elements; $pre, post, asm$ are preconditions, postconditions and assumptions, specified as formulae in $\mathbb{O}$; and $prop$ are domain properties, also specified as formulae. Other features of the ORCAS ACDL are not included here for they play a secondary role or they are elements of the OF.

Figure 2 shows an example of a task specification, Elaborate-query, which refers to the process of transforming a query into one or more queries that are semantically equivalent (synonyms) or related by *information containment* relations among keywords (keywords in the output query that are hyponyms or hypernims of the keywords in the input query). The relation between the input and the output is specified as a collection of postconditions, which in this very simple example is represented by a single Feature Term named Expand-Query.

```
(define (Task :id Elaborate-query)
  (ontologies ISA-Ontology)
  (input (?consult Domain-Query))
  (output (?elab-queries Domain-Queries))
  (postconditions
      Expand-Query))
```

Fig. 2. Task specification using Feature Terms

Figure 3 shows a skill called Query-expansion-with-synonyms, that is able to expand an input query (an instance of Domain-Query), into several new queries (an instance of Domain-Queries) that are semantically equivalents, by replacing some of the keywords in the input query by synonyms taken from a Thesaurus (knowledge role). This kind of transformation produces new queries that are semantically very similar to the original query, thus this skill is a capability suitable to solve the class of problems characterized by the task Elaborate-query.

The synonyms used by this capability to generate new queries are not included in the input, because they are considered knowledge specific of the application domain. Therefore, that capability is independent of any application domain, like medicine or engineering, though it declares the type of knowledge it requires, synonyms. In addition, this capability

```
(define (Skill :id Query-expansion-with-synonyms)
  (ontologies ISA-Ontology)
  (input  (?input-query Domain-Query))
  (output (?new-queries Domain-Queries))
  (postconditions
      Expand-Query-With-Synonyms)
  (knowledge-roles (?thes Thesaurus))
 (assumptions  ACYCLIC-GRAPH))
```

Fig. 3. Capability specification using Feature Terms

imposes an assumption to be verified by a thesaurus in order to be sensibly used: having no cycles (in general a thesaurus may be a graph). Any thesaurus is a potential candidate, but only those thesaurus complying with the assumptions of the capability are allowed. The MeSH thesaurus in the medical domain, for instance, is a non cyclic graph, and thus, it satisfies the assumption of the capability Expand-query-with-synonyms.

To note that the Objet Language is independent of the KMF; the point is that signature and formulae must be refined by the Object Language to yield a precise, computer interpretable meaning. Two examples of expressive languages that could be used as the Object Language are DAML-OIL and OWL. Nevertheless, the choice of a particular Object Language should take into account not only the expressiveness of the formalism; instead, a trade-off between expressiveness and efficiency may be preferred in certain applications. In our implementation of an agent infrastructure we have used a generalization of *First Order* terms called *Feature Terms*[32], and more specifically, a concrete implementation of feature terms as embodied in the NOOS representation language [33]. This formalism organizes concepts into a hierarchy of *sorts*, and represents descriptions and individuals as collections of features (functional relations) called *Feature Terms*.

Matchmaking is the process of verifying wether a matching relation between two components holds: two components "match" if they are substitutable or if one component meets the requirements of the other. Figure 1 shows both the components and the matching relations than can be established among components in the ORCAS KMF. ORCAS extends the usual approach to matchmaking in two ways: first, in addition to the typical *task-capability matching* it includes a *capability-domain matching*; and second, it introduces a Knowledge Configuration process that enables the on-demand design of agent teams according to stated problem requirements. Informally, a task matches a capability if the capability is able to solve the type of problems characterized by the task, and a a capability matches a set of domain-models when the knowledge characterized by those domain-models satisfies the knowledge requirements (assumptions) of the capability. The reader is referred to [8] for a more formal definition of these relations. An important feature of this architecture is that each component can be specified using its own, independent ontology, thus helping to maximize reuse. However, diverse ontologies implies semantic mismatching between component specifications, which turns to be a problem when one needs to compare specifications. As a consequence, the framework must support the specification of ontology mappings between components, and the use of a special kind of components to implement such mappings.

We regard the composition of capabilities as a "bottom-up design problem", that in the ORCAS context is reformulated as: *given a set of problem requirements, find a set of agent capabilities whose combined competence and available knowledge satisfies those requirements*. Our approach to this problem is to use a search process over the space of possible configurations. The result, called *Task-configuration*, is a hierarchical decomposition of a task into subtasks, capabilities bound to tasks and domain models bound to capabilities, in such a way that the resulting Task-configuration satisfies stated problem requirements.

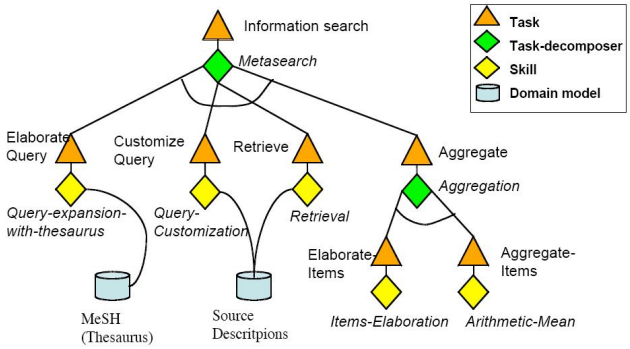


Fig. 4. Task-configuration example

Figure 4 shows an example of a Task-configuration for the *Information-Search* task. This task is decomposed into four tasks by the *Meta-search* task-decomposer: *Elaborate-query*, *Customize-query*, *Retrieve* and *Aggregate*, which is further decomposed by the *Aggregation* capability into two subtasks: *Elaborate-items* and *Aggregate-items*. The example includes some skills requiring domain knowledge: the *Query-expansion-with-thesaurus* requires a thesaurus (e.g. *MeSH*, a medical thesaurus), and the *Retrieval* and *Query-customization* skills require a description of information sources.

B. Operational Framework

The OF describes how a composition of capabilities represented at the knowledge-level can be operationalized by a customized team of agents; in other words, how to form and instruct a team of agents to solve a problem cooperatively and according to a Task-Configuration. The main contributions of this layer are a new model of the Cooperative Problem Solving process, and an extension of the ORCAS KMF to become a full-fledged Agent Capability Description Language.

The ORCAS Cooperative Problem Solving (CPS) process addresses some issues not entirely covered by other frameworks: (1) the need for initial plans to guide the team formation process; (2) the consideration of user preferences and specific problem requirements to constraint the competence of a team; and (3) an external view centered on observable events, rather than an internal view imposing a particular agent architecture. As a result of our work upon these issues we have conceived a new model of the CPS process with four sub-processes (Figure5), namely Problem Specification, Team Design, Team Formation and Teamwork.

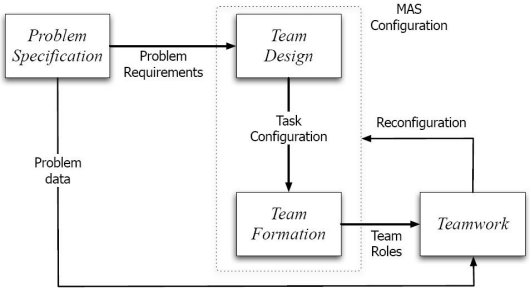


Fig. 5. Overview of the ORCAS Cooperative Problem Solving process.

The result of the Problem Specification process includes input data and problem requirements, including a description of the application domain (a collection of domain models) . The Team Design process aims at finding a Task-configuration satisfying the requirements resulting from the Problem Specification. The Task-configuration is the input to the Team Formation process, which is responsible for allocating tasks to agents and instructing them to solve the problem cooperatively, and according to the given problem requirements. Finally, during the Teamwork process, team members engage in cooperative problem solving, following the instructions received during Team Formation, and thus complying with the requirements of the problem at hand. However, the ORCAS model of the CPS process should not be understood as a fixed sequence of steps. Actually, we have developed several strategies that interleave Team Design, Team Formation, and Teamwork, namely: *Reconfiguration*, *Delayed-Configuration* and *Lazy Configuration* .

- *Reconfiguration* occurs when a task bound to a capability cannot be achieved by neither the selected nor the reserve agents allocated to it. The purpose of reconfiguring a task is to find another capability suitable for that task.
- *Delayed-Configuration* is used to hold the configuration of some task up until some event happens or some information is obtained.
- *Lazy Configuration* is a systematic use of the delayed-configuration: In extremely dynamic environments, configuring a task in advance is not advantageous because it will induce a high frequency of reconfigurations. In this strategy, task are configured on demand, only when strictly required during the Teamwork stage.

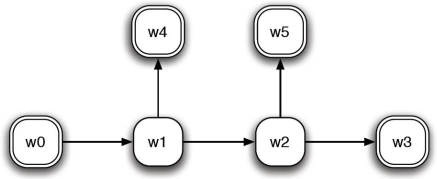
The specification of capabilities according to the ORCAS KMF enables the automated discovery and composition of capabilities, without taking into account neither the communication aspects required to invoke a capability, nor the operational aspects required to coordinate the behavior of several agents. In order to deal with these requirements of agent cooperation, we have extended the ORCAS KMF with two features based on the electronic institutions formalism [6], namely *scenes* and *performative structures*. Scenes are used in ORCAS to specify the *communication* required to invoke a capability, while performative structures are used to specify the *operational description* of a task decomposer.

Below follows a brief description of the concepts used in the electronic institutions formalism.

1) *Agent roles:* standardized patterns of behavior required by agents playing part in given functional relationships.
2) *Dialogic framework:* determines the valid illocutions that can be exchanged among agents, including a vocabulary (ontology) and agent communication language (ACL).
3) *Communication scene:* interaction protocol among a set of agent roles, using the illocutions allowed by a given dialogic framework.
4) *Performative structure:* network of connected scenes that captures the relationships among scenes; a performative structure constrains the paths agents can traverse to move from one scene to another, depending on the roles they are playing.

In ORCAS, a team is defined as a hierarchical organization of team roles resembling a Task-configuration. The agents playing team-roles associated to task decomposers are responsible of delegating subtasks to other agents, receiving partial results, and performing intermediate data processing between subtasks. This simple organization is suited to define all the communication requirements in terms of reusable interaction protocols between two general roles: *Coordinator*, which is adopted by agents applying a task-decomposer, and *Operator*, which is adopted by the agents assigned to the subtasks introduced by a task-decomposer.

Figure 6 depicts a communication example based on the typical request-inform interaction protocol.



1. request (?x Coordinator) (?y Operator) (perform ?team-role ?input)
2. agree !y !x (!team-role !Input)
3. inform !y !x (?team-role ?output)
4. refuse !y !x (!team-role !Input)
5. error !y !x! (team-role !Input)

Fig. 6. Example of a communication scene

An *operational description* describes the control flow among subtasks of a task-decomposer: sequencing of tasks, parallelism, and choice. In ORCAS an operational description is based on the specification of a *performative structure*, but with some distinctive features: in the e-institutions formalism, each scene of a performative structure is instantiated before hand by a concrete communication protocol; in an operational descriptions scenes are instantiated dynamically during the Team Formation process, using as a source the communication protocols shared by the agents that have to cooperate. Each operational description's scene specifies the communication required to solve a subtask, which involves an agent playing the Coordinator role that invokes a capability provided by an agent playing the Operator role. Thus, both the Coordinator and the Operator must adhere to the same protocol in order to communicate, and as a consequence, that protocol must be chosen out of the ones supported by both agents. Since agents are selected dynamically during the Team Formation

process, then also the communication protocols that instantiate an operational description must be chosen dynamically, before Teamwork.

Figure 7 depicts an operational description example for a task-decomposer called Aggregation. This task-decomposer introduces two subtasks: Elaborate-items (EI) and Aggregate-items (AI). Consequently, the operational description has two scenes (in addition to the *Start* and *End* scenes), one for each subtask, and three role variables: x, y, z. x is a Coordinator role, to be played by the agent applying the task-decomposer. x and y are both Operator roles; y participates in EI scene, and z participates in the AI scene. Notice that the Coordinator (x) is the same in both scenes, it enters first EI, and can enter AI only after finishing EI.

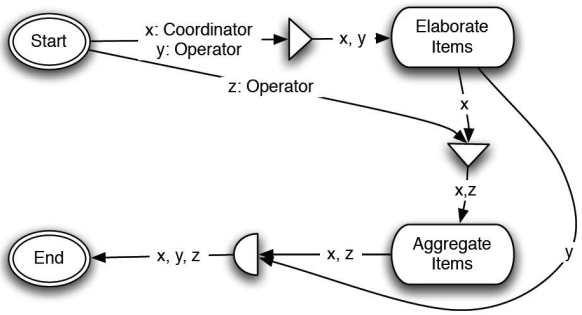


Fig. 7. Example of an operational description

The ORCAS model of a team adheres to the hierarchical decomposition of a task into subtasks embodied by a Task-configuration: starting from a top team-role, each team-role associated to a task-decomposer introduces a set of team-roles subordinated to it. Since each task-decomposer has an operational description, the operational description of a team is modeled as a nested structure of operational descriptions.

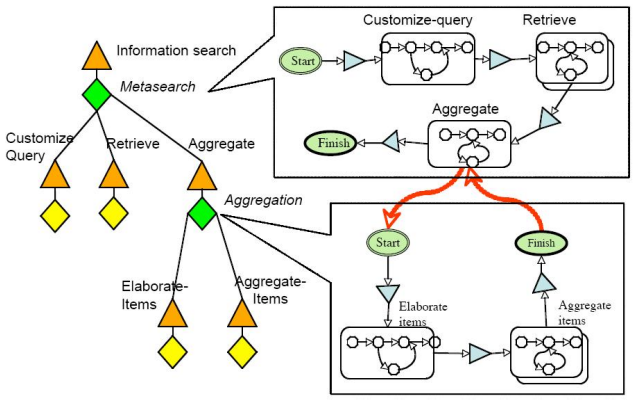


Fig. 8. Teamwork as a nested structure of operational description

Figure 8 shows an example of Teamwork operational description: there is one operational description for the top team-role, called Information Search, with three scenes, one for each subtask: Customize Query, Retrieve and Aggregate. However, the last one has associated a task-decomposer, Aggregation, and thus, it introduces a nested operational description, with two subtasks and their corresponding scenes: Elaborate-Items and Aggregate-Items.

The teamwork process follows the control flow and the communication scenes established by the nested structure of operational descriptions associated to task-decomposers (already instantiated during Team Formation). Each scene within an operational description refers to a communication protocol to be played by two agents, one applying a task-decomposer and playing the Coordinator role, and one assigned to the corresponding subtask playing the Operator role. When an agent playing an Operator role has to apply itself a task-decomposer, it will follow the associated operational description playing itself the Coordinator role. The execution of one operational description does not finish until all the nested operational descriptions are executed.

Each time a new team is formed according to a Task-configuration, a new structure of nested operational descriptions is composed and their scenes instantiated. We regard this structure as a dynamic institution, since it is configured on-the-fly, out of the communication protocols and the operational descriptions supported by the selected team members.

IV. The ORCAS e-Institution

An electronic institution is a "virtual place" designed to support and facilitate certain goals to the human and software agents concurring to that place [34], [35]. Since these goals are achieved by means of the interaction of agents, an e-institution provides the social mediation layer required by agents to achieve a successful interaction: interaction protocols, shared ontologies, communication languages and social behavior rules.

The ORCAS e-Institution acts as a mediation service for both requesters and providers of capabilities, across all the stages of the CPS process. Requesters and providers are ruled by well defined communication protocols, and mediated by institutional agents that reason about application tasks, agent capabilities and problem requirements, using the ORCAS Agent Capability Description Language. The ORCAS e-Institution brings an added value to both requesters and providers of capabilities: on the one hand, the institution frees requesters of finding appropriate providers and presents a single interface to contact and interact with them; on the other hand, the institution offers an advertisement service to capability providers and a mediation service supporting all the stages of the CPS process: Problem Specification, Team Design, Team Formation and Teamwork. The following subsections describe in some detail the main elements of the ORCAS e-Institution.

A. Agent Roles

There are two sorts of agent roles in an e-institution: external roles and internal or institutional roles. In the ORCAS e-Institution, there are two external and three institutional roles. The external roles are: Personal Assistant (PA) and Problem-Solving Agent (PSA), with two subclasses: Coordinator and Operator. The institutional roles are: Librarian, Knowledge-Broker and Team-Broker.

1) *Personal assistant (PA):* It is responsible of mediating between the user and the other agents in the institution, the PA is able to specify problems in terms understood by the Knowledge-Broker, interact with the Team-Broker during the Team Formation process, and start the Teamwork activity once the team is formed.
2) *Librarian:* It holds a dynamic repository or library of component specifications: tasks, capabilities, domain-models and ontologies. This library is dynamically updated by following a registering/deregistering procedure: agents entering the systemregister their capabilities to the Librarian, and deregister when existing it.
3) *Knowledge-Broker:* It takes a library of components and a specification of problem requirements as input, and aims at finding a Task-configuration satisfying those requirements
4) *Team-Broker:* It is responsible of operationalizing a Task-configuration by forming a team of agents with the required competence and knowledge. A team is a group of agents committed and instructed to solve a problem cooperatively, according to a Task-configuration.
5) *Problem-Solving Agent* (PSA): Any agent willing to provide some capabilities to other agents. Problem-Solving Agents can enter and exit the system dynamically, just registering or deregistering their capabilities to the Librarian.

Figure 9 shows the agent roles defined by the ORCAS e-Institution and their relationships, according to the electronic institutions formalism. A *superclass* relationship ($\succeq$) indicates whether a role belongs to a more general class: if role $r \succeq r_0$ then an agent playing r is allowed to play r_0 . A static separation of duties (SSD) between two roles implies that those roles cannot be played simultaneously by the same agent.

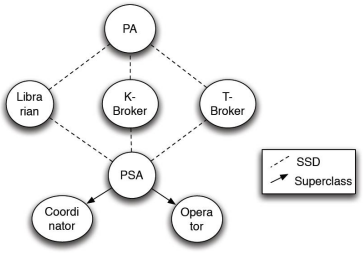


Fig. 9. Agent roles in the ORCAS e-institution

Since the Knowledge Broker, the Team Broker and the Librarian roles are institutional roles, they are preempted of being adopted by an agent playing an external —non institutional— role. This SSD policy protects the institution from being used by external agents to favor themselves in detriment of other agents, thus reinforcing the trust of external agents on the institution. In addition, there are two subclasses of the PSA role: Coordinator and Operator.

- The *Coordinator* role is adopted by a PSA when applying a task-decomposer capability and having to delegate some subtasks to other agents (who play the operator role). The Coordinator is responsible for distributing data among subtask-allocated agents, coordinating them, and performing intermediate data processing
- The *Operator* role is adopted by the agents being delegated a subtask from an agent playing the Coordinator role. Every Operator receive the data they require to solve their tasks from the Coordinator, apply the appropriate capability, and send back the results to the Coordinator.

B. Performative structure

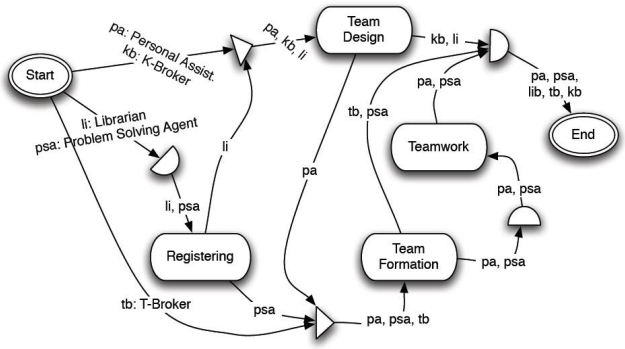


Fig. 10. Performative structure of the ORCAS e-institution

The ORCAS performative structure contains four scenes: Registering, Team Design, Team Formation and Teamwork. Figure 10 depicts the ORCAS e-Institution performative structure, including the four main scenes and the initial (Start) and final (End) scenes. The role-flow policy is represented by the edge labels and transitions between scenes. A PSA has to move first to the Registering scene, and only then can it transition to the Team Formation scene to wait for team-role proposals. The PA must start in the Team Design scene to request the K-Broker for a Task-configuration satisfying the requirements of a problem; afterwards the PA moves to the Team Formation scene to request the Team-Broker for a new team-configuration. Afterwards, the PA moves to the Teamwork scene to request the team-leader of the recently formed team to solve the problem. Finally, the PA provides the team-leader with the input data for the problem at hand, and waits for the results. The different communication scenes are described in the following section, devoting one subsection for each scene.

1) Registering scene: The Registering scene deals with the interaction required for registering and deregistering capabilities to the Librarian. The Registering scene follows a request-inform protocol: when a PSA enters the agent platform, it sends a "register" message to the Librarian, containing the specification of capabilities it is equipped with. Complementarily, agents willing to exit the system must deregister their capabilities, so that the Librarian is continuously updated.

2) Team Design scene: The Team Design scene allows the PA to obtain a Task-configuration satisfying specific problem requirements. A Task-configuration is obtained by a K-Broker applying a Knowledge Configuration process. The Knowledge Configuration process takes a specification of problem requirements and a library of components (tasks, capabilities and domain-models) as input, and produces a Task-configuration

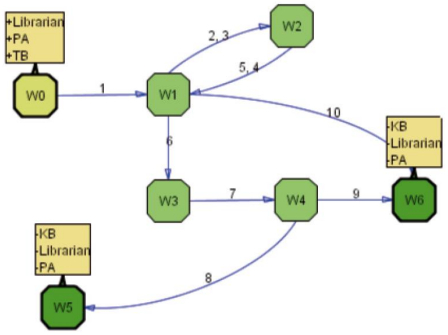
as output. In our implementation, the K-Broker can utilize three search strategies: deep-first search, case-based search, and interactive search.

The requirements of a problem are specified using the same set of features used to specify a task, plus a collection of domain models characterizing the application domain. Figure 11 shows an example of problem requirements from the WIM application, using Feature Terms as the Object Language.

```
(define (Problem-Requirements)
  (ontologies ISA-Ontology)
  (input (?consult Query-Model))
  (postconditions
      Expand-Query-With-Thesaurus
      Non-Exhaustive-Customization
      Aggregate-With-Arithmetic-Mean
      Satisfy-Consult)
   (domain-models
      Medical-Information-Sources
      MeSHThesaurus
      EvidenceBasedMedicine))
```

Fig. 11. Specification of problem requirements using Feature Terms

Figure 12 shows the communication scene for the Team Design scene. The protocol starts (state w_0) with the PA sending a message to the K-Broker. This message (transition 1) is a request to obtain a Task-configuration using the problem requirements included in the content of the message. In the next state (w_1), the K-Broker can either asks the Librarian for the entire library (tr. 2, 4), or it can ask for a partial set of tasks or capabilities satisfying some matching criteria (tr. 3, 5). In the first case the K-Broker gets the entire library in one single interaction, while in the second case the K-Broker obtain component specifications in a more selective way, through several steps. The first step in the Knowledge Configuration process is to choose which task characterizes better the problem at hand. In order to do that, the K-Broker sends the set of tasks matching the initial problem requirements to the PA (w_6), ranking those tasks according to a similarity measure based on past configuration cases. The PA chooses a task and informs the K-Broker (tr. 7) on the chosen task and the preferred search strategy. After receiving the specification of components from the Librarian, the K-Broker starts a Knowledge Configuration process over those specifications. If the K-Broker succeeds in the Knowledge Configuration process, it sends an inform message to the PA, containing the resulting Task-configuration (tr. 8), and the scene ends. Team Design includes another final state that is reached in case of failure, either because the K-Broker cannot find a task satisfying the requirements of the problem (tr. 10), or because the K-Broker cannot obtain a Task-configuration (tr. 9).



1. request (?x PA) (?y KB) (configure ?Problem-Requirements)
2. request (!y KB) (?z Lib) (retrieve-component ?specification)
3. request (!y KB) (?z Lib) (retrieve-library)
4. inform (!z Lib) (!y KB) (found ?components)
5. inform (!z Lib) (!y KB) nothing-found
6. request (!y KB) (!x PA) (choose-task ?task-list)
7. inform (!x A) (!y KB) (configure ?task ?mode))
8. inform (!y KB) (!x PA) (configured ?Task-configuration)
9. failure (!y KB) (!z PA) (partial-configuration ?Task-configuration)
10. failure (!y KB) (!z PA) no-task-found

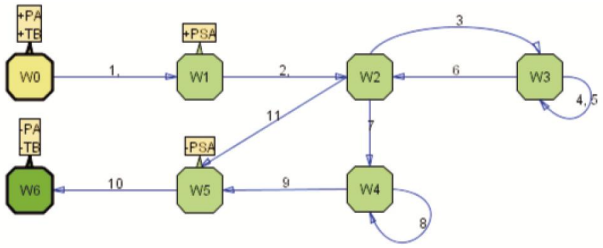
Fig. 12. Specification of the Team Design scene

3) Team Formation scene: The Team Formation scene describes the communication required to form a team of agents that is able to solve a problem according to a Task-configuration. During the Team Formation scene, a team structure composed of team-roles is built according to a Task-configuration, and a group of agents is selected and instructed to play every team-role.

Team Formation comprises three different activities: task allocation, team selection and team instruction. During the task allocation stage, candidate agents are obtained for every team-role. Next, team selection decides the team members to play each team-role out of candidate agents, and keeps other candidate agents in reserve . Finally, the agents participating in Team Formation are informed on the outcome of the selection process, and selected agents are instructed on how to communicate and coordinate with the other team members.

The Team-Broker (T-Broker) is the responsible for guiding the Team Formation process; it mediates between the PA willing to solve a problem and the PSAs providing their capabilities and waiting for requests to join a team. The T-Broker has knowledge about the specification of operational descriptions that is useful to improve the team selection process, and so the performance of Teamwork.

Note that The ORCAS e-Institution is suitable for a wide range of team selection strategies to be embodied within T-Brokers; these strategies belong to the T-Broker internal decision-making, and thus they are not imposed by the ORCAS framework.



1. request (?x PA) (?y TB) (team-formation ?Task-configuration)
2. inform (!y TB) (all PSA) (start-team-formation ?team-id)
3. request (!y TB) (all PSA) (commit ?team-role)
4. accept (?z PSA) (!y TB) (join-team !team-role)
5. refuse (?z PSA) (!y TB) (join-team !team-role)
6. inform (!y TB) (all PSA) (time-out !team-role)
7. inform (!y TB) (all PSA) (start-team-configuration ?team-id))
8. inform (!y TB) (?z PSA) (commit(?team-role)
9. inform (!y TB) (all PSA) (finish-team-configuration ?team-id)
10. inform (!y TB) (all PSA) (finish-team-formation ?team-id)
11. inform (!y TB) (all PSA) (team-failure ?team-id)

Fig. 13. Team Formation Scene

Figure 13 shows the graphical specification of the Team Formation communication scene. There are three roles involved: PA, T-Broker and PSA. The protocol begins with the

PA requesting the T-Broker to form a new team (transition 1) given a Task-configuration. Next, the T-Broker initiates the task allocation activity by informing available agents that a new team formation process is starting (tr. 2). The task-allocation and team selection activities follow an auction-like approach similar to the *contract-net* protocol [36]: team-roles are proposed to PSA agents (tr. 3); PSA agents have to accept (tr. 4) or refuse the proposal before a time-out is reached(tr. 6). If there are no candidate agents for some team-role, the team can not be formed at all. After performing several attempts without success, the T-Broker informs participating agents of a team-failure (tr. 11) and Team Formation is cancelled. If there are candidate agents for all the team-roles, task-allocation succeeds and the T-Broker announces the beginning of the team selection stage (tr. 7). During team selection, the T-Broker has to choose agents to play every team-role ($w4$). Next, the T-Broker informs chosen agents on the team-roles they have to play (tr. 8), including the the following information: a team-role identifier, a task to solve, a capability to apply, the knowledge to use when applying that capability, and optionally, if the capability is a task decomposer, the information required to delegate subtasks to other team-members. Afterwards, the T-Broker informs all the agents that the team-selection process has finished successfully (tr. 9). The scene ends when the T-Broker sends the PA an identifier of the new team as well as the information required to initiate Teamwork (tr. 10).

4) Teamwork scene: The Teamwork scene is performed after a successful Team Formation, and deals with the communication between the PA and the leader of a team (the agent assigned to the top task of the Task-Configuration), who plays the PSA role.

The Teamwork follows a request-inform protocol; the PA sends a "request" message to the team-leader agent. This message contains a team-identifier, an identifier of the team-leader's team role , and the input data for the problem at hand. When the team-leader receives a request from the PA, it checks whether it is committed to the team-role specified in the request. If the target PSA founds that team-role stored in its local memory, it accepts the request, or refuses the request in the opposite case. Next, the PSA checks the type of capability assigned to that team-role in order to figure whether it is a skill or a task-decomposer. If the capability assigned to the team-leader's team-role is a skill, the PSA applies that skill over the input data and give back the result to the PA. In this situation, the team is composed of only one team-role. Otherwise, the capability is a task-decomposer, and the PSA has to consider delegating some subtasks to other agents. This information is provided by a team-role's feature named *subteam*, which specifies the agents selected and keep in reserve for each subtask, and the communication scene to interact with those agents. In this situation, a team is composed of several team-roles, and the Teamwork activity involves one or more performative structures describing the task decomposition control flow,as explained in §III-B (see Figure 8). Recall that there is one performative structure (an operational description) for each task-decomposer in the Task-configuration. As the Teamwork scene proceeds, each task-decomposer expands into a new performative structure that is composed of several communication scenes, one for each subtask being delegated to another agent. Team members adopt the coordinator and operator roles as required, according to whether they are delegating a task to other agent, or a task is being delegated to them, respectively. When team members finish their tasks, they send the results back to their coordinators, which are responsible of obtaining the results of their own tasks, performing intermediate data processing among subtasks, and propagating their own results back to their own coordinators, and so on, until the team-leader obtains the the result of the top task, and sends it to the PA.

V. Conclusions

KMFs propose methodologies, architectures and languages for analyzing, describing and developing knowledge systems [27], [37], [28], [29], [30]. The goal of a KMF is to provide a conceptual model to specify knowledge-based systems in an implementation-independent way, support the engineer in the analysis of a system and facilitate reuse. However, surprisingly KMFs have not been applied to deal with the reuse and interoperation problems in MAS; with some methodological exceptions based on CommonKADs [38], [39]. In this paper, we have described an infrastructure for MAS development and deployment which is based on the integration of the knowledge-modelling stance and the electronic institutions formalism. This infrastructure aims to maximize the reuse of agent capabilities and domain knowledge across different application domains. Next paragraphs discuss similarities and differences between our framework and other frameworks, and sums up the main contributions of our approach.

Some of the first languages for describing agents in open environments were based on logical deduction languages like Prolog; two well known examples are the Interface Communication Language (ICL) used in the Open Agent Architecture [20], [18] to describe agents as goals, and LDL++, used in the InfoSleuth infrastructure [13]. Nonetheless, our way of describing the components of a Multi-Agent System is more similar to LARKS [21], a language used in the RETSINA infrastructure [19] for describing agent capabilities and performing matchmaking. RETSINA is the only infrastructure we know that includes an explicit representation of domain concepts in the specification of capabilities, but in ORCAS there is a more clear decoupling of tasks and capabilities from the application domain. Moreover, while RETSINA relies on Hierarchical Task Network (HTN) planning and scheduling, the ORCAS framework substitutes plans by Task-configurations, and keeps the scheduling activities out of the framework due to its endodeictic nature (belonging to the agent architecture, whereas we adopt an architecturally-neutral approach are focused on the macro, social level). The main difference between ORCAS and other ACDLs is the inclusion of domain-models in addition to tasks and capabilities (equivalent to actions in HTN), which aims at maximizing reuse.

Despite the large research efforts done in the field of Cooperative Problem Solving (CPS), most of the work done falls into one or several stages of the CPS process as presented

in [40], which has four stages: recognition, team formation, planning and execution. The problem solving process starts with an agent willing to solve a task and realizing the potential for cooperation. The process until the task to be solved is decided is usually skipped, assuming that it is already given [41]. Task allocation (part of team formation) requires some kind of preplan describing how to decompose a task into subtasks [42]; two examples are the Planned Team Activity [43] and the Shared Plans [44] approaches. However, often there is a lack of agent-specific algorithms and criteria to build team plans. One approach to team planning is to constrain the competence of the team to satisfy the requirements of the problem at hand. We think that an important issue to address team planning arises from the accountability problem: people may need to understand what happened and why a system alters its response, or may need some predictions on the expected behavior of the system (accountability problem). Some frameworks have addressed the question of the user; for instance the guided team selection approach in [45], the top-down search approach in [46], and the case-based conversational broker described in [47]. In ORCAS, we address the accountability problem by introducing a knowledge-level configuration process before Team formation, what we call Team Design.

Another problem os existing CPS frameworks is the complexity and communication burden of team formation: in large system, team selection may involve an exponential number of possible team combinations, and a blow-out in the number of interactions required to form a team. The ORCAS Team Design stage constraints the competence of a team to satisfy specific problem requirements, which reduces the number of teams to consider during the team formation process, thus considerably reducing the interaction burden.

We adhere to the view of Internet as an open environment where providers and requesters of capabilities interact to solve problems cooperatively. This view of Internet as a distributed computational platform is in spirit the same of the Semantic Web initiative, in particular, our view of agent capabilities shows some aspects that are closer to some Semantic Web frameworks than to existing Agent Capability Description Languages. From the Semantic Web approach, building an application is basically a process of composing, connecting and verifying the properties of Semantic Web Services (SWS) in a way that resembles our compositional approach to Team Design. There are, however, two major differences between SWS frameworks and ORCAS. On the one hand, ORCAS agents are autonomous entities that can decide to accept or to refuse a request, while services are reactive, passive entities which are directly invoked by; therefore, instead of a centralized composition of services, we view the composition of capabilities as a negotiation process among autonomous agents. On the other hand, the ORCAS ACDL is domain independent, intended to maximize reuse, while Web Services frameworks ignore this issue, since they are domain dependent by nature (a Web service is associated to some concrete domain, like the weather of a specific country in a weather forecasting service).

In conclusion, although ORCAS advances some steps forward towards open MAS, there is still much work to do. For instance, the mappings between ontologies used to specify different components, such as capabilities and domain models, are currently predefined and hardwired in the ORCAS e-Institution. In order to improve the reusability of components developed by third parties, the platform should provide tools supporting the development and implementation of ontology mappings. Other aspects of the platform deserving future work are the protocols for handling dynamic events such as agent failure, since they are now dealing only with a reduced number of situations.

REFERENCES

[1] M. Klein, "The Challenge: Enabling Robust Open Multi-Agent Systems," 2000. [Online]. Available: citeseer.ist.psu.edu/380348.html

[2] M. Luck, P. McBurney, and C. Preist, *Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*. AgentLink, 2003.

[3] K. Decker, K. Sycara, and M. Williamson, "Middle-agents for the internet," in *Proceedings the 15th International Joint Conference on Artificial Intelligence*, 1997, pp. 578–583.

[4] K. P. Sycara, M. Klusch, S. Widoff, and J. Lu, "Dynamic service matchmaking among agents in open information environments," *SIGMOD*, vol. 28, no. 1, pp. 47–53, 1999.

[5] M. Gómez, *Open, Reusable and Configurable Multi-Agent Systems: A Knowledge-Modelling Approach*, ser. Monografies de l'Institut d'Investigació en Intel.ligència Artificial. Spanish National Research Council, 2004, vol. 23.

[6] M. Esteva, J. A. Rodriguez, C. Sierra, P. Garcia, and J. L. Arcos, "On the formal specifications of electronic institutions," in *Agent-mediated Electronic commerce. The European AgentLink Perspective*, ser. Lecture Notes in Artificial Intelligence, vol. 1991, 2001, pp. 126–147.

[7] M. Gómez and J. M. Abasolo, "A general framework for meta-search based on query weighting and numerical aggregation operators," in *Intelligent Systems for Information Processing: From Representation to Applications*. Elsevier Science, 2003, pp. 129–140.

[8] M. Gómez and E. Plaza, "Extending matchmaking to maximize capability reuse," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'04, Vol. 1*, 2004, pp. 144–151.

[9] ——, "Integrating knowledge modeling and multi-agent systems," in *Proceedings of the AAAI Fall Symposium on Agents and the Semantic Web*. AAAI Press, 2005, pp. 25–29.

[10] K. Decker, M. Williamson, and K. Sycara, "Matchmaking and brokering," in *Proceedings of the 2nd International Conference in Multi-Agent Systems*, 1996. [Online]. Available: citeseer.nj.nec.com/decker96matchmaking.html

[11] T. Erickson, "An agent-based framework for interoperability," in *Software Agents*, J. M. Bradshaw, Ed. AAAI Press, 1996.

[12] M. R. Genesereth and S. P. Ketchpel, "Software agents," *Communications of the ACM*, vol. 37, no. 7, 1997. [Online]. Available: citeseer.nj.nec.com/genesereth94software.html

[13] M. Nodine, W. Bohrer, and A. Ngu, "Semantic brokering over dynamic heterogeneous data sources in infosleuth," in *ICDE*, 1999, pp. 358–365.

[14] W. P. Birmingham, E. H. Durfee, T. Mullen, and M. P. Wellman, "The distributed agent architecture of the university of michigan digital library (extended abstract)," in *AAAI Spring Symposium on Information Gathering*, 1995. [Online]. Available: citeseer.nj.nec.com/birmingham95distributed.html

[15] J. M. Vidal, T. Mullen, P. Weinstein, and E. H. Durfee, "The UMDL service market society," in *Proceedings of the Second International Conference on Autonomous Agents*, 1998. [Online]. Available: http://www.acm.org/pubs/articles/proceedings/ai/280765/p475-vidal/p475-vidal.pdf

[16] E. H. Durfee, T. Mullen, S. Park, J. M. Vidal, and P. Weinstein, "The dynamics of the UMDL service market society," in *Cooperative Information Agents II*, ser. Lecture Notes in Artificial Intelligence, M. Klusch and G. Weiss, Eds. Springer, 1998, pp. 55–78.

[17] J. M. Vidal and E. H. Durfee, "Task planning agents in the UMDL," in *Proceedings of the Fourth International Conference on Information and Knowledge Management (CIKM) Workshop on Intelligent Information Agents*, 1995. [Online]. Available: citeseer.nj.nec.com/vidal95task.html

[18] A. Cheyer and D. Martin, "The open agent architecture," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 4, no. 1, pp. 143–148, March 2001, oAA.
[19] K. P. Sycara, M. Paolucci, M. V. Velsen, and J. A. Giampapa, "The RETSINA MAS infrastructure," Robotics Institute, Carnegie Mellon University, Tech. Rep., 2001.
[20] D. L. Martin, A. J. Cheyer, and D. B. Moran, "The open agent architecture: A framework for building distributed software systems," *Applied Artificial Intelligence*, vol. 13, no. 1-2, pp. 91–128, January-March 1999, oAA.
[21] K. P. Sycara, S. Widoff, M. Klusch, and J. Lu, "Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace," *Autonomous Agents and Multi-Agent Systems*, vol. 5, pp. 173–203, 2002.
[22] N. Guarino, "Semantic matching: Formal ontological distinctions for information organization, extraction, and integration," in *Summer School on Information Extraction*, M. Pazienza, Ed. Springer Verlag, 1997, pp. 139–170. [Online]. Available: citeseer.nj.nec.com/guarino97semantic.html
[23] A. W. Brown and K. C. Wallnau, "Engineering of component-based systems," in *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute*. IEEE Computer Society Press, 1996, pp. 7–15.
[24] F. M. Hafedh Mili and A. Mili, "Reusing software: Issues and research directions," *Software Engineering*, vol. 21, no. 6, pp. 528–562, 1995. [Online]. Available: citeseer.nj.nec.com/mili95reusing.html
[25] R. J. Hall, "Generalized behavior-based retrieval," in *Proceedings of 15th International Conference on Software Engineering*, 1993, pp. 371–380.
[26] Z. Zhang, "Enhancing component reuse using search techniques," in *Proceedings of 23rd conference on Information System Research in Scandinavia*, 2000.
[27] L. Steels, "Components of expertise," *AI Magazine*, vol. 11, no. 2, pp. 28–49, 1990.
[28] J. McDermott, "Toward a taxonomy of problem-solving methods," in *Automating Knowledge Acquisition for Expert Systems*, S. Marcus, Ed. Kluwer Academic, 1988, pp. 225–256.
[29] A. Schreiber, B. J. Wielinga, J. Ackermans, W. Van de Velde, and R. D. Hoog, "Commonkads: A comprehensive methodology for kbs development," *IEEE Expert*, vol. 9, no. 6, pp. 28–37, 1994.
[30] D. Fensel, V. Benjamins, E. Motta, and B. Wielinga, "UPML: A framework for knowledge system reuse," in *International Joint Conference on AI*, 1999, pp. 16–23.
[31] A. Newell, "The knowledge level," *Artificial Intelligence*, vol. 28, no. 2, pp. 87–127, 1982.
[32] E. Plaza, "Cases as terms: A feature term approach to the structured representation of cases," in *ICCBR*, 1995, pp. 265–276.
[33] J. L. Arcos, "The noos representation language," Monografies del IIIA, Universitat Politècnica de Catalunya, 1997.
[34] P. Noriega, "Agent-mediated auctions: The fish-market metaphor," Ph.D. dissertation, Universitat Autnoma de Barcelona, 1997.
[35] J. A. Rodríguez-Aguilar, "On the design and construction of agent-mediated electronic institutions," Ph.D. dissertation, Universitat Autnoma de Barcelona, 1997.
[36] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Transactions on Computers*, vol. C-29, no. 12, pp. 1104–1113, December 1940.
[37] B. Chandrasekaran, "Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design," *IEEE Expert*, vol. 1, pp. 23–30, 1986.
[38] C. A. Iglesias, M. Garijo, J. Centeno-Gonzalez, and J. R. Velasco, "Analysis and design of multiagent systems using MAS-common KADS," in *Agent Theories, Architectures, and Languages*, 1997, pp. 313–327. [Online]. Available: citeseer.nj.nec.com/iglesias98analysis.html
[39] N. Glaser, "Contribution to knowledge modelling in a multi-agent framework," Ph.D. dissertation, L'Université Henri Poincaré, Nancy I, France, 1996.
[40] M. Wooldridge and N. R. Jennings, "Towards a theory of cooperative problem solving," in *Proceedings Modelling Autonomous Agents in a Multi-Agent World*, 1994, pp. 15–26. [Online]. Available: citeseer.nj.nec.com/wooldridge96towards.html
[41] ——, "The cooperative problem-solving process," *Journal of Logic and Computation*, vol. 9, no. 4, pp. 563–592, 1999. [Online]. Available: citeseer.nj.nec.com/wooldridge99cooperative.html
[42] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 165–200, 1998. [Online]. Available: citeseer.nj.nec.com/shehory98methods.html
[43] E. Sonenberg, G. Tidhar, E. Werner, D. Kinny, M. Ljungberg, and A. Rao, "Planned team activity," in *Articial Social Systems*, ser. Lecture Notes in Articial Intelligence, 1994, pp. 227–256.
[44] B. J. Grosz and S. Kraus, "Collaborative plans for complex group action," *Artificial Intelligence*, vol. 86, no. 2, pp. 269–357, 1996.
[45] G. Tidhar, A. Rao, and E. Sonenberg, "Guided team selection," in *In Proceedings of the 2nd International Conference on Multi-agent Systems (ICMAS-96)*, 1996.
[46] B. J. Clement and E. H. Durfee, "Top-down search for coordinating the hierarchical plans or multiple agents," in *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, O. Etzioni, J. P. Muller, and J. M. Bradshaw, Eds. Seattle, WA, USA: ACM Press, 1999, pp. 252–259.
[47] H. Munoz-Avila, D. W. Aha, L. Breslow, and D. S. Nau, "HICAP: An interactive case-based planning architecture and its application to noncombatant evacuation operations," in *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*. AAAI Press, 1999, pp. 879–885.

ACKNOWLEDGEMENTS

The authors would like to thank the Spanish Council for Scientific Research for their support. This work was supported by MID-CBR (TIN 2006-15140-C03-01) project.

Mario Gomez is a research fellow for the Department of Computing Science at the University of Aberdeen. He holds Licentiate degrees in Computer Science and Psychology. He was a research assistant at the Barcelona Artificial Intelligence Research Institute (IIIA) for 4 years, and completed a Ph.D in Computer Science by the Autonomous University of Barcelona in 2004. Afterwards, he moved to Madrid to take a position as visitant professor, and has been giving lectures on Artificial Intelligence related subjects for two years. As part of his research, he has participated in several national, european and international projects, and has authored a book and several scientific papers in a number of fields, ranging from knowledge modeling and information retrieval to multi-agent systems and computational models of trust. Some of his technical contributions to those fields have been awarded with national and international prices.

Enric Plaza holds a Ph.D. in Computer Science by the Technical University of Catalonia (UPC) and is Researcher of the CSIC (Spanish Council for Scientific Research) at the Barcelona Artificial Intelligence Research Institute (IIIA) since 1988. He has worked on knowledge acquisition, case-based reasoning, and machine learning in a dozen of European and Spanish projects. He has chaired three international conferences on A.I. fields and has authored over 120 scientific papers. His research is now focused on new techniques for case-based reasoning, learning and argumentation in the framework of open multi-agent systems. He is member of a dozen program committees annually for international conferences, plus other conferences and workshops. He is ECCAI fellow and has served as Chairman of the Board of Trustees of the ACIA (Catalan Association for Artificial Intelligence) during four years and frequently publishes articles for AI popularization.