Chapter 5

The Operational Framework

This chapter describes an operational model of the Cooperative Problem-Solving process for Multi-Agent Systems, and which is the role played by the Knowledge-Modelling Framework within this process.

5.1 Introduction

The Operational Framework describes a mapping from the concepts in the Knowledge-Modelling Framework to concepts from Multi-Agent Systems and Cooperative Problem Solving. Specifically, the Operational Framework describes how a task-configuration —obtained by the Knowledge Configuration process— can be operationalized by forming a customized team of problem solving agents on-demand, according to stated problem requirements. Moreover, the Operational Framework describes also the communication and the coordination mechanisms required by agents to carry over the Cooperative Problem-Solving process according to a task-configuration.

The Operational Framework extends the Knowledge Modelling Framework to develop a full-fledged Agent Capability Description Language (ACDL). The ORCAS ACDL is used in open Multi-Agent Systems by *requesters* willing to solve a problem, *providers* of problem solving capabilities, and *middle agents* responsible for mediating between them (e.g. brokers and matchmakers).

1. *Requesters* use the ACDL to put a query Q describing the type of problem to be solved, characterized by a task (the application task T_0), a collection of domain models characterizing the application domain (Q_{dm}), and other requirements of the problem (preconditions and postconditions).

2. *Providers* use the ACDL to describe the tasks they can solved and the capabilities they are equipped with.

3. *Middle agents* are used in open MAS to mediate between requesters and providers. Middle agents are responsible for locating appropriate providers

for a given request and facilitating the interaction between requester and providers. For instance, a broker frees the requester of knowing the details required to invoke and interact with each specific provider, so the requester only have to know how to interact with the middle agent and not with each potential provider.

In our approach to the Cooperative Problem Solving (CPS) process and also in our particular implementation of an open agent platform supporting the CPS process, the ACDL is used for the following activities: (1) in the automatic design of agent teams at the knowledge level, as a configuration of components (tasks, agent capabilities and domain models) satisfying stated problem requirements; (2) to guide the team formation process according to the configuration of components at the knowledge-level; and (3) to coordinate the behavior of team members during the teamwork.

This chapter is divided as follows: we start with an overview of the Cooperative Problem-Solving (CPS) process in §5.2; the extensions of the ORCAS KMF to become an Agent Capability Description Language (ACDL) are described in §5.4; the main concepts of the ORCAS team model are defined in §5.3; the Team Formation process is described in §5.5; the ORCAS model of Teamwork is presented next, in §5.6; and the Chapter ends with a brief discussion of some extensions of the CPS process in §5.7 .

5.2 The Cooperative Problem-Solving process

The view on Multi-Agent Systems as decoupled networks of autonomous entities is usually associated to a distributed model of expertise: A MAS is regarded as a collection of specialized agents with complementary skills.

Most of the research done in the field of *Cooperative Problem Solving* (CPS) (e.g. the models based in the Contract Net protocol [Smith, 1940] or derived from the Generalized Global Planning approach [Durfee, 1988]) falls into one or more or the stages of a general model of the Cooperative Problem-Solving process as presented in [Wooldridge and Jennings, 1994], which consists of four stages: *recognition*, *team formation*, *planning* and *execution*.

1. *Recognition*: the CPS begins when some agent recognizes the potential for cooperative action. This recognition may come about because an agent has a goal that it does not have either the ability or the resources to achieve on its own, or else because the agent prefers a cooperative solution in expectation of getting some benefit.

2. *Team formation*: during this stage, the agent that recognized the potential for the cooperative action at stage (1) requests assistance. If this stage is successful, it will end with a group of agents having some kind of nominal commitment to collective action.

3. *Plan negotiation*: during this stage, the agents proceed to negotiate a joint plan which they believe will achieve the desired goal.

4. *Execution (Team action)*: During this stage, the newly agreed plan of joint action is executed by the agents, which maintain a close-knit relationship throughout. This relationship is defined by a convention, which every agent follows.

Actually, these four stages are *iterative*, in that if one stage fails, agents may return to previous stages. Although the proposers of this model believe that most instances of CPS exhibit these stages in some form (either explicitly or implicitly), they stressed that the model is idealized. In other words, there are cases that the model cannot account for [Wooldridge and Jennings, 1999].

In concordance with the proposers of this model, we think it is well suited for a number of situations, but is not adequate for others (the reader is referred [Wooldridge and Jennings, 1994, Wooldridge and Jennings, 1999] to for a deeper understanding of the model. Since team formation is not guided by a preplan to achieve the overall goal, but is just a commitment to joint action, then neither the agents joining a team (committing to carry on joint action) are guaranteed to play one role in the team once a plan was decided at the subsequent planning stage, nor the resulting team assures that a global plan can be found.

This uncertainty may be not a problem if the MAS is composed of quite homogeneous problem solving agents with a common range of skills. In such homogeneous (in term of functionalities) agent societies, the very same agent could potentially occupy many different positions within a team, and thus the possibility of forming a successful team grows up. This approach has been adopted mostly when cooperation is defined [Norman, 1994] as acting with others for a common purpose and a common benefit where the purpose should be motivated by an intention to act together —a joint intention [Cohen and Levesque, 1990, Levesque, 1990, Cohen and Levesque, 1991], and resolved by a commitment to joint activity [Bratman, 1992, Jennings, 1993].

This class of cooperation relying on motivational attitudes is sometimes called collaboration [Wilsker, 1996, Grosz and Kraus, 1996] to differentiate it from other classes of cooperation. It is not surprising then that implemented frameworks inspired by a collaborative approach to cooperation are usually applied in scenarios where the team-oriented agents are homogeneous and their roles typically represent either authority relations, such as military rank, or high-level capability descriptions [Tambe, 1997, Pynadath et al., 1999].

Moreover, if the MAS is composed of specialized agents equipped with very specific capabilities, the following bizarre situations may happen: first, agents that have joined the team may not be needed after the plan negotiation have finished; second, team roles may remain unassigned after the plan negotiation; and third, the plan can remain incomplete. In order to prevent these types of failure (specially the second and third cases, which suppose the team is not able to achieve the global goal at all), specialized agents should be able to reason about goals and plans in order to acquire team goals, to identify the roles that contribute to the fulfillment of a team goal, and to decide whether to commit to a particular team role. Team formation and also plan negotiation without a initial guide on the types of tasks to be solved and the capabilities required, can provoke an exponential grow in the number of teams and plans to be considered, and a blow-out in communication overload as the size of the population grows up or the complexity of the team goal increases.

Other researchers have explored the utility of using an initial plan to guide the team formation process. The SharedPlans theory [Grosz and Kraus, 1996] and the frameworks based on it, e.g. [Giampapa and Sycara, 2002], have emphasized the need for a common, high-level team model that allows agents to understand all requirements for plans that might achieve a team goal. Team plans are used by agents to acquire goals, to identify roles and to relate individual goals to team goals. An initiator of a cooperative activity can use an initial team plan to know the functionalities or competencies required to achieve the overall goals.

An initial plan allows the initiator of the team formation process to know which are the subgoals and (optionally) the actions or capabilities required to achieve each subgoal. Therefore, the initiator can use the initial plan to guide the team formation process [Tidhar et al., 1996]. An initial team plan allows the initiator of the cooperative activity to contact only with the agents holding the required capabilities, thus increasing the possibility of success, and reducing the complexity of both the team formation and the plan negotiation processes.

Another issue concerning the CPS process and related with the idea of guided team formation is that of problem requirements: how to design a team according to the requirements of a specific problem, rather than selecting a plan according to a fixed task. The CPS model in [Wooldridge and Jennings, 1999] does not explicitly address the utility of constraining the competence or behavior of a team to satisfy stated problem requirements or comply to user preferences. Recent initiatives in MAS planning have introduced case-based [Munoz-Avila et al., 1999] and conversational planners to build the initial plans to be adopted by a team [Giampapa and Sycara, 2001].

Moreover, this model of the CPS process devises an internal perspective on agents, in which the agent's internal state is used as the basis for evaluation. Concerning this issue, though we recognize there are well founded reasons to adopt an internal perspective, we think a external view has also some advantages and, in particular, it is more appropriate for open systems because it avoids imposing a model of the agent architecture to external agent developers. We are developing our model under this assumption, and thus we try to impose minimal requirements on the internals of agents willing to participate in a CPS process beyond the use of the ORCAS Knowledge Modelling Ontology. These requirements consist basically of a shared communication language and a set of interaction protocols enabling team action. Summarizing, we avoid imposing neither a specific agent architecture, nor a model of cooperation based on mental attitudes. Instead, an external view centered on the observable patterns of agent communication and commitment is adopted here.

Summarizing, there are some issues related to Cooperative MAS not covered by the general model of the CPS process in [Wooldridge and Jennings, 1999]: (1) the need for an initial plan to guide the team formation process; (2) the consideration of the user preferences and specific problem requirements constraining the composition of teams from a set of possible to a set of allowed teams; and (3) the use of a external view centered on observable events like the illocutionary acts rather than a internal view imposing a particular agent model or architecture.

As a result of our work upon these issues we have conceived a new model of the CPS process that is based on the use of a Knowledge-Modelling Framework to describe a MAS at an abstract-level, and introduces a Knowledge-Configuration process as a mechanism to design the behavior of an agent team by building an initial team "plan" satisfying stated problem requirements. Such initial plans can be used to drive the team formation process and to coordinate agent behavior during the teamwork.

Now we are going to carry out a more detailed review of the main activities involved in the CPS process —team formation, planning and execution— in order to compare our approach to the planning based approaches that are commonly used in cooperative MAS.

Team formation is defined as the process of selecting a group of agents that have complimentary skills to achieve a given goal [Tidhar et al., 1996]. Typically, team formation has been divided in two activities: selecting a group of agents that will attempt to achieve the team goal, and selecting a combination of actions that agents must perform to achieve the goal [Levesque, 1990, Cohen and Levesque, 1991, Rao and Georgeff, 1995, Grosz and Kraus, 1996, Tambe, 1997], also approached as a plan negotiation process [Wooldridge and Jennings, 1999]. This combination of actions is typically described as a sequence of actions or a *plan* [Georgeff and Lansky, 1987, Bratman, 1988, Rao et al., 1992, Grosz and Kraus, 1993, Sonenberg et al., 1994, Grosz et al., 1999, Tate, 1998]. In many approaches there are partial plans hold by different agents that must negotiate a given plan until consensus is reached about an agreed global plan [Ephrati and Rosenschein, 1996]. In other approaches the plan is built by merging individual plans —like the approaches based on the SharedPlans theory, for instance in the RETSINA teamwork model [Giampapa and Sycara, 2002], — until all the tasks required by the global plan are assigned. While some approaches start the selection of team members without an initial plan, other approaches [Tidhar et al., 1996, Giampapa and Sycara, 2001] use it to drive the team formation process, even when the planning process is distributed among agents [Clement and Durfee, 1999].

One of the preferred approaches to represent plans in MAS is based on using some kind of hierarchical plans, like Hierarchical Task Networks (HTN) [Erol et al., 1994, Erol, 1995]. The way in which an HTN structure decomposes a task into subtasks is similar to the way tasks are decomposed by a task-decomposer in the ORCAS KMF. Nevertheless, ORCAS configuration structures are not oriented towards planning algorithms; instead, ORCAS structures are designed to maximize the reuse of agent capabilities by decoupling the description of capabilities from the application domain through the use of independent ontologies to describe them both. In spite of these differences, ORCAS configuration structures are used as recipes about the tasks (or goals) to achieve and the capabilities (or actions) required to achieve them, and thus they play the same role than a plan.

In ORCAS there are three types of structure concerning planning: task-decomposition schemas (subtasks introduced by a task-decomposer and ordered by the operational description); *configuration schemas* resulting of binding a task-decomposition schema to a collection of capabilities (one capability is bound to each subtask); and *task-configurations*, which are composed of interrelated configuration schemas (see §4.4.1).

An approach to multi-agent planning is that of obtaining a global plan by merging individual, (usually partial) plans. In ORCAS the individual plans are partial (agents have a local view), and are represented by task-decomposers and configuration-schemas, which are two ways of representing how to achieve a task by decomposing it into subtasks. A configuration-schema is a more specific representation than a task-decomposer, since the former includes the capabilities required to solve each task, while a task-decomposer informs only about the subtasks of a decomposition, but not about the capabilities required to achieve them.

The role of a global team plan in ORCAS is played by a *task-configuration*, since a task-configuration is used as a recipe about the actions (the capabilities) required from team members to achieve the goals of the team (represented in ORCAS by an application task plus some extra problem requirements).

Task-configurations obtained at the Knowledge Configuration process are used to guide the team formation process and to coordinate team members during the Teamwork process: an agent willing to start a cooperative activity uses such an initial plan to know which are the tasks to be solved, which are the capabilities to apply, and which is the knowledge to be used by the selected capabilities. The agent responsible for coordinating the team formation process can use the information provided by a task-configuration to select the agents that are potential candidates to join the team (though a yellow pages service): only agents with the required capabilities are considered, thus the number of possible teams to be considered is reduced and the communication requirements decrease. In addition, the Knowledge-Configuration process ensures that a task-configuration satisfies stated problem requirements, therefore the teams that are formed complying to a task-configuration are guaranteed to satisfy the requirements of the problem too.

Teamwork is the process carried out by a team of agents in order to achieve a global team goal. In ORCAS the team goal is initially represented by the application task and subsequently refined by a task-configuration. A task-configuration contains a specification of tasks and subtasks to be achieved (a hierarchical task decomposition structure), the competence required to solve those tasks (the capabilities), and the specification of the application domain (the domain-models).

The global behavior of a team during the Teamwork process is guided by the task-configuration, since team members commit to solve the application task by applying the capabilities and domain-models specified in the task-configuration.

To move from the knowledge level to the operational level, we have to match concepts from a task-configuration to agent concepts. Our proposal is to define a one-to-one mapping between tasks in a task-configuration and roles played within a team, that we call *team-roles*. There is one team-role for each task, where each team-role contains the information an agent needs to solve a task in the context of a team. During the execution stage of the Cooperative Problem-Solving process, team members have to cooperate with other team mates in order to achieve the overall goal. Specifically, we are interested here in a hierarchical, top-down style of cooperation, since it fits well into the hierarchical structure of a task-configuration. This style of cooperation is based on a task-delegation mechanism.

A motivation for separating the Knowledge Configuration process from the team formation process is to exploit the fact that the specification of agent capabilities remains stable throughout long periods of time, whereas there are dynamic aspects of the system or the environment that change very quickly and are not deterministic (i.e. workload, network traffic, system failures). As a consequence, the Knowledge Configuration process aims to explore the utility of a configuration of capabilities in terms of their static and abstract descriptions, keeping such aspects isolated from the the dynamic and operational aspects involved in the CPS process (e.g. the workload of a problem solving agent), which are managed during the Team Formation and Teamwork processes.

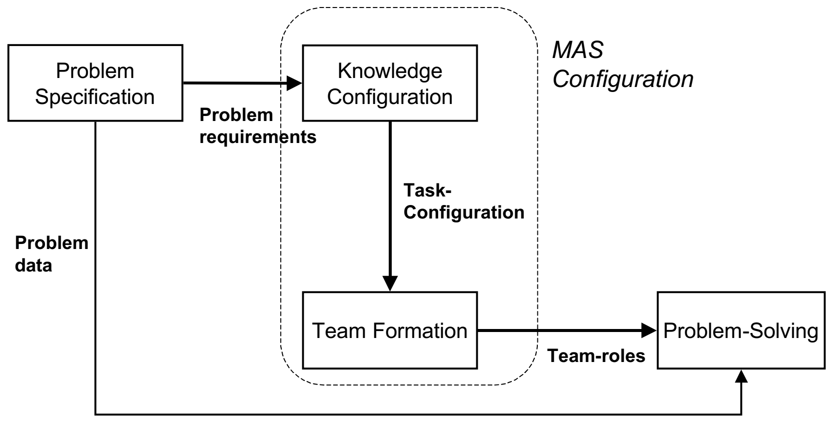


Figure 5.1: The ORCAS model of the Cooperative Problem-Solving process

Figure 5.1 shows the main elements of the Cooperative Problem Solving process and the main relations between them. The CPS process starts with a Problem Specification process in which the problem requirements and the problem data are supplied. The Knowledge Configuration process takes the problem requirements and a library of components as input and builds a task-configuration. The Team Formation process uses a task-configuration to form a new team of agents satisfying the conditions established by that task-configuration (i.e. agents commit to the conditions enforced on the tasks allocated to it, its team roles). The outcome of the Team Formation process is a configuration of the team expressed as a collection of interrelated team roles to be played by the selected team members, and a group of specific agents assigned to these roles. Finally, during the Teamwork process the new team solves the problem at hand according to the task-configuration obtained at the Knowledge Configuration process and a specification of the input data provided at the Problem Specification process.

Although the CPS process as showed in Figure 5.1 seems to follow a sequential control flow, this model is just a simplification that is intended to highlight the cornerstones of the model. Once the different subprocesses of the CPS process have been explained we are in a better position to extend the model in order to deal with more complex situations. Actually, we have added mechanisms for interleaving all the stages of the Cooperative Problem-Solving process. This feature will allow a team to be reconfigured in order to deal with dynamic conditions and events encountered during the different stages of the CPS process. Some extensions of this model are described in §
refsec:extensions.

5.3 Team model

The ORCAS team model is defined upon concepts from the Knowledge Modelling Framework, specifically, ORCAS teams are defined according to an abstract model of teamwork based on the structure and the meaning of a task-configuration.

A *team* is a group of agents that commits to solve a problem in a cooperative way, according to a task-configuration. A *team* is composed of agents that have complimentary capabilities to achieve a global goal and are assigned to different *roles* within the team. A team has the goal of solving a specific problem by using a task-configuration as a recipe about the task to be solved (specifying the global goal), a decomposition of the main task into subtasks, the capabilities to be applied, and the domain knowledge required.

As explained in §4.4, the result of the Knowledge Configuration process is a *task-configuration*, a hierarchical structure where nodes are triplets consisting of a task, a capability bound to the task, and optionally some domain-model satisfying the assumptions of the capability. In order to understand the way a task-configuration is operationalized by a team of agents, it is necessary to establish a mapping between the concepts involved in a task-configuration and concepts from Multi-Agent Systems.

A task-configuration in the ORCAS team model plays the role of a team plan, since it is used as a recipe of actions to achieve a global goal. The configured task represents a global team goal and a plan to achieve that goal. Specifically, tasks represent goals and subgoals; skills represent primitive, non decomposable actions to achieve goals; and task decomposers play the role of sub-plans in decomposing a task (a goal) into subtasks (subgoals). Multiple capabilities allowed for the same task represent alternative ways of solving a problem, or alternative ways of decomposing a problem into subproblems. In addition, a task-configuration contain the domain models satisfying the knowledge requirements of the selected capabilities.

In order to map elements from the KMF to elements of agent-based teamwork, we introduce the notion of *team-role*. A team-role defines the functions assigned to a position within the team. We establish a one-to-one correspondence between tasks and team-roles: there is a team-role for each task within a task-configuration. A task-configuration follows a hierarchical task decomposition structure, and a team in ORCAS is based on a task-configuration; consequently, teams are also organized hierarchically, as a nested structure of teams and subteams (Figure 5.2).

Team-roles define the competence required by the different members of a team in order to achieve a team goal. A team-role includes all the information required to solve one of the tasks of a task-configuration, that is to say, a task, a capability bound to it, and optionally a set of domain models required by the selected capability. When the capability bound to a task is a task-decomposer, a team role is defined for the task being decomposed, and a new team-role is created for each subtask. The agent playing the role of the task-decomposer acts as the *coordinator* (supervisor or leader) of the team-roles assigned to each subtask, that are in some sense "subordinated" to the coordinator, though the precise nature of the relationship between the task-decomposer role and its subordinated team-roles may vary according to features like the degree of agent autonomy or the degree of openness of the MAS.

Figure 5.2 shows an example of a team modelled as a hierarchy of team-roles that is organized as a nested structure of teams, and the straightforward mapping of tasks and capabilities from a task-configuration to team-roles. There is a team-role for each task in the task-configuration, and there is one team for each task-decomposer bound to a task. Each team consist of a "coordinator" team-role responsible for the task being decomposed, plus a set of "subordinated" team-roles, one per subtask. In Figure 5.2, the Team-role 1 (TR1) is assigned to the information-search task, and has to to apply the meta-search task-decomposer capability, which introduces four subtasks. Therefore, TR1 has to coordinate the activity of their subordinated team-roles, one for each subtasks: elaborate-query (TR2), customize-query (TR3), retrieve (TR4) and aggregate (TR5). Moreover, since TR5 is itself assigned to task-decomposer (aggregation), it has to play the coordinator role to interact with their own subordinated team-roles, those associated to their two subtasks, namely elaborate-items (TR6) and aggregate-items (TR7).

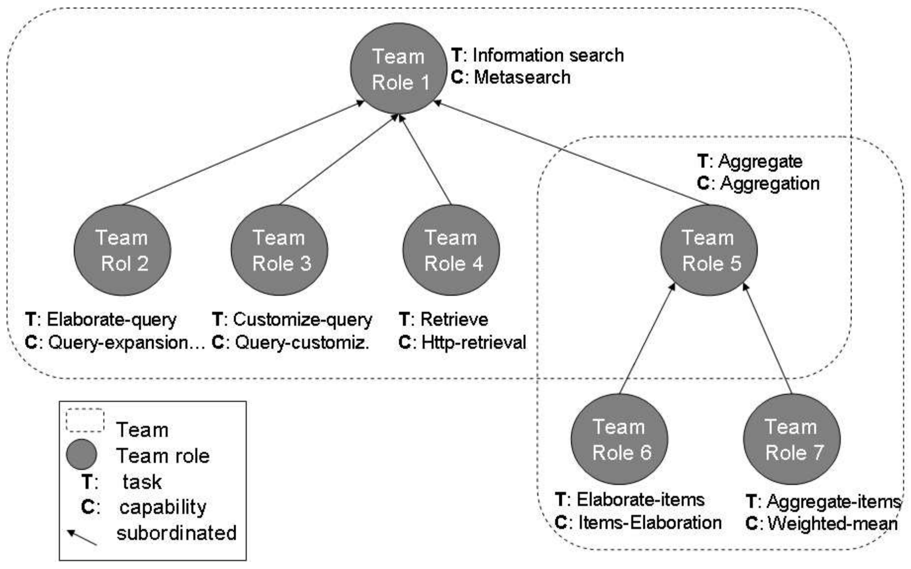


Figure 5.2: Model of a team as a hierarchical team-roles structure

5.3.1 Team-roles and team-components

A *team-role* describes the functional, operational and pragmatic aspects required of an agent to occupy a position within a team. The basic information included within a team-role is a task to be solved (a goal), the capability to be applied (suitable for the goal task), and the communication elements (language and protocols) to be used for communicating with the agent playing that team-role. In addition, if the capability is a task-decomposer, then the team-role can include information about team members selected for solving each subtask, the communication elements required to delegate each subtask to the selected agent, and optionally a group of agents to keep in reserve.

Formally, a team-role is defined as follows:

Definition 5.1 ***(Team-Role)*** *A team-role is defined as a tuple*

$$\pi = \langle R, I, T, C, M, Com, S, A_S, A_R \rangle$$

where

- *R is a unique team-role identifier,*
- *I is a unique team identifier,*
- *T is a task,*
- *C is a capability,*
- *M is a set of domain-models,*

- *Com is a specification of communication requirements,*
- *A_S is a set of selected agents,*
- *A_R is a set of reserve agents,*
- *S is a subteam, specified as a set of team-components.*

The *team-role identifier* (R) is required to unambiguously identify the position of a team-role in the team hierarchy. The name of the task is not enough to identify a team-role because the task could appear multiple times within a task-configuration. The *team identifier* (I) is required because one agent can participate in multiple teams simultaneously. A team-role includes also the name of the *task* to be achieved (T), the name of the *capability* selected (C) to solve that task (according to a task-capability binding in the task-configuration), and optionally a set of domain-models (M) satisfying the knowledge requirements of the selected capability. In order to instantiate a team model with specific team-members, a team-role includes also two slots to specify a set of agents selected for it (A_S), and a set of reserve agents (A_R). Moreover, a team-role specifies the *communication* (Com) model to be used by both the requester or coordinator of R, and the agent assigned to R.

Finally, a team-role includes a *subteam* feature to be filled only when the team-role's capability (C) is a task-decomposer. A *subteam* is specified as a set of *team-components*, where each team-component holds information about a team-roles associated to one subtask. A team-component is defined as follows:

Definition 5.2 ***(Team-Component)*** *A team-component is defined as a tuple*

$$\xi = \langle R, T, A_S, A_R, Com \rangle$$

where

- *R is a unique team-role identifier,*
- *T is a task*
- *A_S is a set of selected agents*
- *A_R is a set of reserve agents*
- *Com is a specification of communication requirements*

A team-component is defined for each subtask introduced by a task-decomposer. The *team-role identifier* (R) determines the precise position of the team-component in the team hierarchy. There is a set of agents selected (A_S) to carry out the team-role, and there is a set of agents to keep in reserve (A_R) for the case that some of the selected agents fail during the Teamwork process. Finally, a team-component includes a specification of the *communication* (Com) required to interact with the agent playing the team-component's team-role (R).

Figure 5.3 shows an example of a team-role assigned to a task-decomposer with two subtasks. Team-role 5 (TR5) is assigned to the Aggregate task, that is bound to the Aggregation capability. This capability is a task-decomposer that introduces two subtasks: Elaborate-Items and Aggregate-Items. Therefore, the TR5 team-role has a subteam with two team-components, TR6 and TR7, assigned to the Elaborate-Items and the Aggregate-Items tasks respectively.

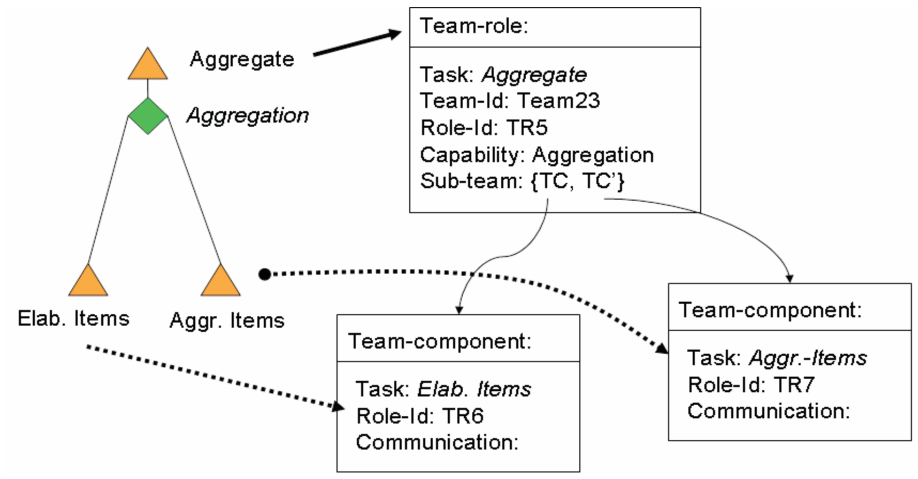


Figure 5.3: From tasks to team-roles and team-components

We note π and Π as a team-role and the set of all the team-roles, and ξ and Ξ as a team-component and the set of all the team-components. We can define now a team as a structure made of interrelated team-roles and team-components, but first we define the *subordinated relation* $\mathbb{S}$ among team-roles as follows:

Definition 5.3 ***(Subordinated)***

$$\mathbb{S}(\pi, \pi') \Leftrightarrow \exists \xi^i \in \pi_S \mid \xi^i_R = \pi'$$

where

- $\pi, \pi' \in \Pi$ *are team-roles;*
- $\pi_S \subseteq \Xi$ *is the subteam of* π *(a set of team-components);*
- $\xi^i \in \Xi$ *is the i-th element of* π_S
- $\xi^i_R \in \Pi$ *is the team-role associated to* ξ^i

Briefly, a team-role is subordinated to another if the first team-role is bound to a team-component contained in the subteam of the second team-role.

Noting $\mathbb{S}^*$ the closure of $\mathbb{S}$ we can now define a *team* as follows:

Definition 5.4 ***(Team)*** *A team is defined for a team-role and a task-configuration*

$$Team(\pi^0, Conf(\kappa)) = \{\pi \in \Pi | \mathbb{S}^*(\pi^0, \pi) \wedge (head(\kappa) = \pi^0_T)\}$$

where

- $\pi^0 \in \Pi$ *is a team leader's team-role, the only team-role that is not subordinated to another;*
- $Conf(\kappa)$ *is a task-configuration,*
- $head(\kappa)$ *is the root task of the task-configuration* ($Conf(\kappa)$)*,*
- *and* π^0_T *is the task assigned to the team leader's team-role.*

A *team* is a collection of interrelated team-roles, starting from the team-leader π, that is assigned to the root task of a task-configuration. The ORCAS team model provides an abstract view of the competence required by a group of agents to solve a global problem. Teams are instantiated during the Team Formation process (§5.5) by selecting a set of agents to play each team-role, and a set of agents to keep in reserve.

Team-roles are used during the Team Formation and the Teamwork processes to interchange information among agents. During the Team Formation process, team-roles are used as advertisements or proposals to join a team: team-roles are used to inform potential team members of the tasks to be solved, the capabilities to apply, the knowledge to use, and optionally the terms of commitment and pragmatic issues. Team-roles can also be used by agents to send counter-proposals during the Team Formation process. After finishing the task allocation and the agent selection process, team-roles are used to inform agents about the result of the process.

During the Teamwork process, team-roles are used by team members to know every thing they need to achieve his tasks, to delegate subtasks to other agents, to cooperate with other agents when requested, and even to rescue from unexpected situations like agent failures preempting a task to be achieved by the selected agent, or communication problems avoiding an agent to send or receive messages from other team member. A team-role has the information required by an agent to play a team-role: the name of the task to be solved, the capability to apply, the knowledge to use, and optionally a set of team-components (a subteam) with the information required to delegate the team-role's subtasks to other team members. This information includes the identifiers of subordinated team-roles (those to whom delegate some subtask), a set of selected agents to play each subordinated team-role and a set of agents to keep in reserve for each one, but it does not include information about the capability that should be used by each subordinated team-role.

A team member willing to carry out the task assigned to a team-role during the Teamwork process has to check whether the capability to be applied is a task-decomposer or a skill. On the one hand, if the capability assigned to a team-role is a skill, the agent will engage in communication with the requester. The interaction protocol, the vocabulary and the language to be used will be defined also within a team-role, using the communication feature, as described in §5.4.2. On the other hand, an agent willing to apply a task-decomposer capability has to engage in communication with a requester, just like an agent applying a skill, but in addition the agent must know which agents are assigned to the task-decomposer's subtasks, so as to request them to carry out their subtasks. The communication to be used so as to delegate a task to another agent is also provided within a team-role structure.

However, an agent applying a task-decomposer does not need to know which capability will be used by the different components of a sub-team, because this information will be sent to those agents separately. In few words, each member of the team knows what to do, which tasks to delegate and to whom, but a team member does not know the precise way a task he delegates to another agent will be solved by it. The last statement is not mandatory, is just a question of information economy, but can be modified to accommodate better to specific situations.

5.4 The **ORCAS** Agent Capability Description Language

The notion of an *Agent Capability Description Language* (ACDL) has been introduced recently [Sycara et al., 1999a] as a key element in enabling MAS interoperation in open environments. An ACDL is a shared language that allows heterogeneous agents to coordinate effectively across distributed networks. Sometimes, capabilities are referred as "services" and, consequently, an ACDL can alternatively be called an Agent Service Description Language (ASDL).

In the literature, an ACDL is defined as a language to describe both agent advertisements and requests, and is primarily used by middle agents (e.g. brokers and matchmakers) to pair service-requests with service-providing agents that meet the requirements of the request [Sycara et al., 1999b, Sycara et al., 1999a].

Some desirable features for such a language are *expressiveness*, *efficient reasoning* and *easy use*:

- *Expressiveness*: the language should be expressive enough to represent not only data and knowledge, but also the meaning of a capability. Agent capabilities should be described at an abstract rather than implementation level.

- *Efficient reasoning*: inferences on descriptions written in this language should be supported. Automatic reasoning and comparison on the descriptions should be both feasible and efficient.

- *Easy use*: descriptions should not only be easy to read and understand, but also easy to write. The language should support the specification of knowledge requirements (in order to link capabilities to domain knowledge) and the use of ontologies for specifying agent capabilities in a way that favors reuse.

Another important aspect to take into account for designing an ACDL is the idea of enriching capability descriptions with semantic information [Paolucci et al., 2002]. Semantic markup, which is based on the use of shared ontologies [Guarino, 1997a], improves the matchmaking process and facilitates interoperation.

Although the ORCAS KMF satisfies these requirements, we think an ACDL should bring support to some activities involved in MAS interoperation beyond the discovery of capability providers. An ACDL should facilitate the automation of the following activities, namely discovery, execution, composition and interoperation of capabilities:

Automatic capability discovering (matchmaking): This activity takes the specification of a request and looks for capabilities that are able to satisfy such request. This activity involves the automatic location of capabilities that adhere to requested constraints, which is usually described as a matchmaking process between the request and a library or repository of capabilities (typically hold by a middle agent). An ACDL must allow capability providers to advertise their capabilities to the matchmaker or *yellow pages* service in order to become available for automatic capability discovery. In ORCAS the discovery of capabilities satisfying a problem specification is also achieved through a matchmaking process (§4.2.2), but the ORCAS Knowledge Configuration process goes beyond this requirements and introduces the idea of configuring (designing) a complete MAS-based application (a configured team) that satisfies a specification of stated problem requirements, rather than finding appropriate providers of capabilities suitable for a a single task. The aspects of a capability description required for these activities are the functional descriptions described in the previous chapter: the interface (inputs and outputs) and the competence (preconditions and postconditions), plus the aspects of a capability used to filter out those capabilities which knowledge requirements are not fulfilled.

Automatic capability execution (communication): Having selected a capability, the process of enacting or executing it. Agents should be able to interpret the description of a capability to understand what input is necessary to execute a capability, what information will be returned and which are the effects or postconditions that will hold after applying the capability. In addition, the requester of a capability must know the communication protocol, the communication language and the data format required by the provider of the capability in order to sucessfully communicate with it. Summing up, an ACDL should provide declarative descriptions of both the interfaces and the communication requirements required for executing agent capabilities on request. In ORCAS these aspects of a capability are partially fullfilled by the already described functional aspects of a capability (inputs and outputs, competence, and knowledge requirements), but the communication aspects (interaction protocol and language, and data format) has not been described yet, though we have aforementioned them when talking about the communication feature of a capability (§4.2.1). These aspects are anticipated in the ORCAS KMF, where two features of a capability have been earmarked for further extension of the KM-Ontology into a fullfledged ACDL: the *communication* and the *operational description*. A proposal for describing these features is introduced later, in §5.4.2 and §5.4.3.

Automatic capability composition (configuration): In order to achieve more complex tasks, capabilities may be combined or aggregated to achieve complex goals that existing capabilities cannot achieve alone. This process may require a combination of matchmaking, capability selection among alternative candidates, and verification of wether the aggregated functionality satisfies the specification of a high-level goal. In ORCAS capabilities are composed during the configuration of a task at the Knowledge Configuration process, using the matching relations introduced in the Knowledge Modelling Framework (§4.2.2), and ensuring that the resulting configuration satisfies the stated problem requirements.

Automatic capability interoperation (coordination): Multiples agents involved in solving a task by applying a composed capability to solve a global task should interoperate between them. Sharing an agent communication language, a common vocabulary and the same interaction protocols are necessary, but not sufficient for cooperation to succeed. In addition to the communication aspects, interoperation among specialized agents during teamwork has to deal with the coordination of agent activities according to the sequencing of tasks and possible task-dependencies. In ORCAS, the information required to coordinate agent behaviors during teamwork is provided by the *operational-description* of the capabilities composing a task-configuration.

The functional description of a capability as provided in the Knowledge-Modelling Framework (Figure 4.8) enables the automated discovery and composition of capabilities (configuration). Nonetheless, the execution of capabilities and the interoperation of multiple agents during teamwork are not supported by the functional description of a capability.

In order to deal with these activities, we have included two extra features to characterize a capability: the *communication* and the *operational description*. Since we keep the knowledge-level aspects of a capability separated from the operational aspects, we avoid including them within the description of a capability in the KMF. This decision allows the ORCAS KMF to be used across different implementations of a MAS and even different types of computational system, like semantic Web services.

The usual approach to overcome the interoperability problems arising in open MAS is to assume a common language and interaction protocols, while the operational aspects of a capability are assumed to be part of the own agent control and thus, they are not declared by an agent when registering its capabilities to a middle agent. While the former elements must be shared by a group of agents in order to work together, open agent environments are encouraged to support more flexible approaches. Therefore, we think the use of declarative descriptions of the communicative and the operational aspects of a capability will support a more flexible architecture where cooperating agents can choose from their repertoire of languages and protocols those that are more appropriate at the moment.

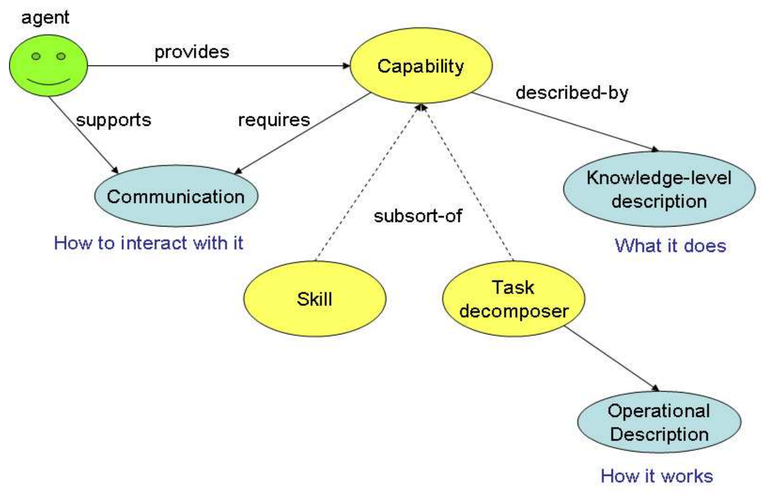


Figure 5.4: Main elements of the ORCAS ACDL concerning capabilities

Figure 5.4 shows the main elements of the ORCAS ACDL. A capability is provided by an agent, and can be either a skill or a task-decomposer. The knowledge-level description of a capability as provided in the Knowledge Modelling Framework (§4.2.1) answers the question "what a capability does?"; that is to say, the KMF provides a functional view of agent capabilities. When a capability is task-decomposer, the ORCAS ACDL provides an operational-description specifying how a task is decomposed into subtasks. In addition, the ORCAS ACDL introduces a communication model that specifies how to interact with an agent so as to request him to apply a capability. The communication model describes the language and the interaction protocol supported by an agent in providing his services to other agents. In general, the same communication model can be used for different capabilities, but some capabilities may not suit some communication models. Therefore, communication models and capabilities are described independently, so as to maximize reuse of both communication models and capabilities. Nonetheless, agents keep the link between the capabilities they provide and the communication models they support over each capability.

The functional aspects of capability are covered by the KMF, which is focused only on those aspects required by the Knowledge-Configuration process. Therefore, those aspects involving agent concepts, like the communication and the operational description, are subjects of the Operational Framework. Specifically, they are presented as two subsections (§5.4.2 and §5.4.3) of the ORCAS ACDL section. concerning the agent approach (e.g. the communication) have been left undefined at the KMF and are addressed at the Operational Framework. But prior to describe these elements in detail we are going to overview the formalism used to specify them: *electronic institutions* [Esteva et al., 2001, Esteva et al., 2002b].

5.4.1 Electronic Institutions

We have stated previously (Chapter 1) our decision of adopting a social, macro-view of Multi-Agent Systems. In particular, we adopt the formal approach of electronic institutions [Esteva et al., 2001, Esteva et al., 2002b] to specifying open agent societies, which is based on a computational metaphor of human institutions.

Human institutions are places where people meet to achieve some goals following specific procedures, e.g. auction houses, parliaments, stock exchange markets, etc. Intuitively, the notion of agent-mediated institution or electronic institution proposes a sort of virtual places where agents interact according to explicit conventions. The institution is the responsible for defining the rules of the game, to enforce them and impose the penalties in case of violation.

An electronic institution, or e-Institution, is a "virtual place" designed to support and facilitate certain goals to be achieved by human and software agents concurring to that place. Since these goals are achieved by means of the interaction of agents, an e-institution must provide the social mediation layer required to achieve a successful interaction: interaction protocols, shared ontologies, communication languages and social behavior rules. An example of such an institution is an Auction House. An Auction House has *institutional agents*, those agents (like the auctioneer) that manage the tasks required for the institution to exist; but Auction Houses are open: they allow other agents (buyers and sellers) to "enter" that place in order to achieve their own goals.

The main goal of the e-Institutions approach is the specification and automatic generation of infrastructures for open agent organizations. Electronic institutions are architecturally-neutral with respect to agents, focused on the macro-level (social view) of agents, and not in their micro-level (internal view).

We are interested on using the concepts proposed by the e-Institutions approach as a way to specify the interaction and coordination needs of teamwork without imposing neither a specific agent architecture, nor a mentalistic theory of cooperation . In electronic institutions, all agent interactions can be reduced to illocutions. Therefore, accountability is expressible in terms of how illocutions are constrained, or what characteristics can be predicated and tested on illocution utterance, and on illocution reception.

A more precise definition of electronic institutions follows [IIIA, 2003]:

> An electronic institution is the computational realization of a set of explicit, possibly enforceable restrictions imposed on a collection of dialogical agent types that concur in space and time to perform a finite repertoire of satisfiable actions.

This definition assumes that agents are "dialogical entities" that interact with other agents within a multi-agent context which is relatively static in ontological terms. We can assume also that agents exhibit rational behavior by engaging in dialogical exchanges, i.e. that agent interactions are systematically linked to illocutions that are comprehensible to participants and refer to a basic shared ontology, and that the exchanges can be (externally) construed as rational. Moreover, the institution is the real depositary of the ontology and interaction conventions used by the participating agents.

Dialogical agents are entities that are capable of expressing illocutions and react to illocutions addressed to them and, furthermore, only illocutions (and the contextual effects of their associated actions, e.g. commitments to sell a good) constitute observable agent behavior. Individual agents may have other capabilities —perception, intentions, beliefs, etc.—, but we assume that as long as agents interact within the institution, only illocutions are perceivable by other agents (and the meaning and conditions of satisfaction of the associated actions can be objectively established and accounted for within the shared context). Moreover, agents within an institution can only utter illocutions that are consistent with the "role" they are playing. Definitively, the e-Institutions approach is social or exodeitic [Singh, 1998], focused on the macro-level view of Multi-Agent Systems and not on any particular agent architecture.

An e-institution is modelled with the following components [Noriega, 1997, Rodríguez-Aguilar, 1997, Esteva et al., 2001]:

1. *Agent roles:* Agents are the players in an electronic institution, interacting by the exchange of speech acts, whereas roles are standardized patterns of behavior required by agents playing part in given functional relationships. Any agent within an electronic institution is required to adopt some role.

2. *Dialogic framework:* A dialogic framework determines the valid illocutions that can be exchanged among the agents participating in an electronic institution. In dialogical institutions, agents interact through speech acts, thus the institution establishes the acceptable speech acts by defining the ontological elements (the vocabulary) and the agent communication language (ACL). By sharing a dialogic framework, an electronic institution enables heterogeneous agents to meaningfully interact with others.

3. *Communication scenes:* Interaction protocols are articulated through agent group meetings called scenes. A scene defines an interaction protocol among a set of agent roles using a specific dialogic framework. A scene is a formal specification of a pattern of structured communication which constrains the possible patterns of dialogues that can be used by the participating agents adopting one of the roles in the scene (agents have to adopt some role in order to participate).

4. *Performative structure:* A performative structure is a network of connected scenes that captures the relationships among scenes. The specification of a performative structure contains a description of how the different

agent roles can move from one scene to another. Furthermore, agents may participate in different scenes, playing different roles at the same time, or engage in multiple instances of the same scene simultaneously.

5. *Normative rules:* Agent actions may have consequences that either limit or enlarge its subsequent acting possibilities. Such consequences are specified through normative rules, which impose obligations to the agents and affect their possible paths across the performative structure.

ORCAS approaches open agent organizations as virtual, agent based institutions composed of heterogeneous agents playing different roles and interacting by means of speech acts. However, while electronic institutions have been proposed as a way to describe static or predefined organizations, we are rather interested on a more dynamic approach in which the institution is built on-the-fly by putting existing pieces together. While tasks and capabilities where combined during the Knowledge Configuration process to compose a task-configuration, scenes and performative structures are combined and integrated during the Team Formation process. The result is an electronic institution ad-hoc, which provides the communication and the coordination elements required for a team to achieve a global problem according to stated problem requirements.

The following subsections describe the elements required to communicate and coordinate with other agents during the Teamwork process. We will introduce the required notions from electronic institutions and then the way we use those concepts in ORCAS.

5.4.2 Communication

The *communication* aspects of a capability describe the elements required to interact with an agent providing that capability. Interaction is required, basically, to send input data to an agent willing to execute a capability, and to receive back the output produced by the application of a capability from another agent. ORCAS agents are dialogical entities that communicate using speech acts or illocutions; more specifically, we use elements of the electronic-institutions approach to describe the communication aspects associated to a capability. The communication requirements of a capability are specified as scenes using some dialogic framework. A dialogic framework contains the elements for the construction of the communication language, expressions used within the capability communication scenes. Scenes are dialogical patterns of interactions based on the illocutions and vocabulary defined by the dialogic framework. In ORCAS scenes are used to describe the interaction protocols supported by an agent providing some capability. Therefore, scenes are bound to some capabilities within the context of an agent equipped with that capability. The idea here is that of individual agents equipped with a set of capabilities and a set of scenes supported by each capability.

Figure 5.5 shows the Communication sort, used to describe the *communication* features of a capability: a set of *scenes* describing different interaction

Communication
scenes $\rightarrow$ set-of *Scene*
dialogic-frameworks $\rightarrow$ set-of *Dialogic-Framework*

Figure 5.5: The Communication and Communication-scenes sorts

protocols supported by an agent, and a set of dialogic-frameworks describing the vocabulary and the languages supported by an agent.

The following subsections describe the two elements of the electronic institutions formalism used in ORCAS to describe the communication aspects of a capability: dialogic frameworks and scenes.

Dialogic framework

In open environments agents can be endowed with its own inner language and ontology. In order to allow agents to successfully interact with other agents their languages and ontologies must be put in relation. For this purpose, the electronic institutions approach establishes that agents must share a *dialogic framework* that contains the elements for the construction of the communication language expressions that are required within the institution or within a specific scene. By sharing a dialogic framework, heterogeneous agents can exchange knowledge by means of illocutionary acts.

The electronic institutions formalism defines a dialogic framework as follows [Esteva, 1997]:

Definition 5.5 ***(Dialogic Framework)*** *A dialogic framework is defined as a tuple $DF = \langle O, L, I, R_I, R_E, R_S \rangle$, where*

- *O stands for an ontology (vocabulary);*
- *L stands for a content language to express the information exchanged between agents;*
- *I is the a of illocutionary particles;*
- *R_I is a set of internal roles;*
- *R_E is a set of external roles;*
- *R_S is a set of relationships over roles.*

The dialogic framework determines the valid illocutions (I) that can be exchanged between the participants. In order to do so, an ontology (O) that fixes what are the possible values for the concepts in a given domain is defined, e.g goods, participants, locations, etc. Moreover, the dialogic framework defines which are the roles that participating agents may play within the institution or within a particular communication scene the dialogic framework is bound to.

Each role defines a pattern of behavior within the institution. Roles allow to abstract from the individuals agents participating in the electronic institution. This feature is specially important in open environments in which the identity of the agents that could participate in the institution is not known in advance. Furthermore, in open environments agents may change over time, since new agents may join the institution and agents already in the institution may come to leave. For this reason, all the actions that can be done within an institution are associated to roles, and not to individual agents. Intuitively we can think of roles as agent types characterized by a set of actions allowed for that type. For instance, within an auction, an agent playing the buyer role is capable of submitting bids, while the agent playing the auctioneer role can offer goods at auction. In order to take part in an electronic institution, an agent is obliged to adopt some role(s). An agent playing a given role must conform to the pattern of behavior attached to that particular role. However, all agents adopting a specific role are guaranteed to have the same rights, duties and opportunities.

A dialogic framework distinguishes internal roles (R_I) from external roles (R_E). Internal roles define the roles to be played by staff agents, which are equivalent the employees of a human institution. Those agents are in charge of guaranteeing the correct functioning of an institution. For instance, an auctioneer is in charge of auctioning goods following the specified protocol and the buyer admitter is in charge of guaranteeing that only buyers satisfying the admission conditions are allowed to participate.

Two types of agent relationships over roles can be specified, namely: *superclass* and *static separation of duties* (SSD). *Superclass* relationships indicate whether a role belongs to a more general class. If a role r is a superclass of another role r' ($r \succeq r'$), then an agent playing r is enabled to play r'. However, since agents can play several roles at the same time, role relationships standing for conflict of interests must be defined with the purpose of protecting the institution against an agent's malicious behavior. For instance, in an auction house the auctioneer and the buyer roles are mutually exclusive. A static separation of duties policy is defined to avoid two mutually exclusive roles of being authorized to the same agent. The static separation of duties is defined as the relation $ssd \subseteq Roles \times Roles$. A pair $(r, r') \in ssd$ denotes that r, r' cannot be authorized to the same agent. See [Esteva et al., 2001] for an enumeration of the requirements for the *ssd* relation and some inferred properties.

The content language (L) allows for the encoding of the knowledge to be exchanged among agents using the vocabulary offered by the ontology. The propositions built with the aid of the content language are embedded into an "outer language", the communication language(CL), which expresses the intentions of the utterance by means of the illocutionary particles.

Expressions in the communication language are constructed as formulae of the type $(\iota\,(\alpha_i\,\pi_i)\,(\beta)\,\varphi\,\tau)$ where ι is an *illocutionary particle*, α_i is a term which can be either an agent variable or an agent identifier, π_i is a term which can be either a role variable or a role identifier, β represents the addressee(s) of the message (which can be an agent or a group of agents), φ is an expression in the *content language* and τ is a term which can be either a time variable or a time constant. The CL allows to express that an illocution is addressed to an agent, to all the agents playing a role or to all the agents in the scene.

Notice that a dialogic framework determines the ontological elements and the valid elements for constructing expressions in the communication language. Thus a dialogic framework must be regarded as a necessary ingredient to specify scenes.

Using dialogic-frameworks in ORCAS: the Teamwork ontology

Given a capability C, one can think that some of the ontological elements required to interact with the agent providing C are those used to specify C, like the signature-elements used to specify the input and the output. Nonetheless, we have preferred to keep the knowledge-level elements away from the operational and communication aspects so as to maximize the reuse of these elements too. In order to do that, we have decided to specify the communication of a capability independently of other features of the capability: we specify the communication aspects using generic, easy to reuse concepts, like the notion of input and output, and not in terms of a particular type of input (a signature-element).

Since the ORCAS framework aims to maximize reuse (of both capabilities and communication elements), thus we try to impose as few requirements as possible to agents willing to cooperate, thus we have imposed a minimum set of concepts to be understood by agents in order to participate in an ORCAS e-Institution. These concepts should be shared by all the members of a team in order to cooperate, thus they are explicitly represented as an ontology. This ontology consist of the previously presented notions of Team-role (§5.1) and Team-component (§5.2), plus some concepts relating the different types of messages that can be exchanged during the Teamwork process. A basic model of communication for teamwork includes the following type of messages:

- *Perform*: requests to solve the tasks associated to a specific team-role.
- *Result*: messages containing the results of having performed some task.
- *Done*: messages to confirm that some request has been achieved
- *Refusal*: messages to inform that the requested petition won't be carried on.
- *Failure*: messages to inform of some failure occurred while performing the tasks associated to a team-role.

Figure 5.6 shows the basic concepts included in the Teamwork ontology, and the features characterizing each concept. These concepts and their features are defined as sorts in the Teamwork ontology, and are used within scenes to constrain the type of messages allowed in the following way: only messages complying with the illocutionary schemas defined by the scene are permitted. For example, if an illocutionary schema specifies that a message should have a

Teamwork ontology	
Team-role	Definition 5.1
Team-component	Definition 5.2
Perform	team-id:Symbol, team-role:Symbol, input-data:Signature-element
Result	team-id:Symbol, team-role:Symbol, output-data:Signature-element
Done	team-id:Symbol, team-role:Symbol
Refusal	team-id:Symbol, team-role:Symbol, reason:Any
Failure	team-id:Symbol, team-role:Symbol, error:Any

Figure 5.6: Basic Teamwork concepts

content of the type *Perform*, then only messages with that type of content would be allowed. The sort Any subsumes any other sort and is used to allow further specialization by developers, for instance, the reason for a refusal or the error code when informing about a failure.

Capabilities should be specified independently of other capabilities in order to maximize their reuse and facilitate their specification by third party agent developers. In the general case, agent developers do not know a priori the tasks that could be achieved by an agent capability, since teams are formed on-demand according to specified problem requirements, and thus the same capability could be used to solve different tasks (as far as the the capability is suitable for the task, as defined by a task-capability matching relation) or the same task in the context of a different task-decomposition. As a consequence, the team roles an agent can play using a capability are not known in advance. Therefore, the roles used to specify the communication scenes of a capability cannot be specified in terms of specific team-roles.

Our approach to overcome the former difficulty is to specify the communication aspects of a capability in terms of abstract, generic roles, rather than specific team-roles. As already explained, ORCAS teams are hierarchically organized according to task-configurations, and thus the teamwork itself can be easily coordinated using a hierarchical communication style. There are agents decomposing a task into subtasks and requesting other agents to solve some of the subtasks. An agent that applies a task-decomposer capability to solve a task is responsible for delegating subtasks to other agents, receiving the results, and performing intermediate data processing between subtasks. In such a scenario, we can establish an abstract communication model with two basic roles:

1. the *coordinator* role is adopted by an agent willing to decompose a task into subtasks, requesting other agents to carry on the different subtasks in an appropriate order, receiving the different results, and obtaining the final result of the task; and

2. the *operator* role, one the other side, is adopted by the agent having to perform a task on demand, using the data provided by another agent that acts as coordinator, and having to bring the result back to the coordinator.

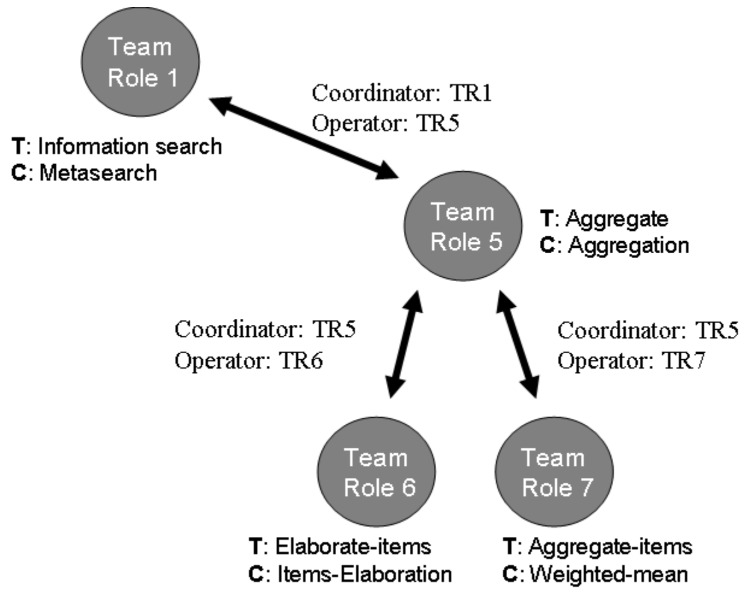


Figure 5.7: Example of team-role relations and role-policy for Teamwork

The operator and the coordinator roles are defined as keeping a static separation of duties (SSD) relationship, so as to avoid an agent to adopt both roles simultaneously within the same scene. However, the same agent can act as coordinator and operator simultaneously, but playing those roles in different scenes. This situation occurs when a team-role is neither the root neither a leave in the team-roles hierarchy. The agent playing such a team-role has to adopt the operator role to communicate with the agent assigned the task-decomposer on top of it (one level above in the team hierarchy). Nonetheless, in order to finish its task, this agent has to communicate with other agents assigned to its own subtasks, and adopting the coordinator role itself. Figure 5.7 shows an example of such a situation. The agent playing Team-role 5 (TR5) has to act as operator to communicate with TR1. Notice that TR5 (the agent playing TR5) has to solve the task Aggregate using the input data received from TR1, and send the result back to TR1; but in order to do that, TR5 must delegate its own subtasks (elaborate-items and aggregate-items) to its subordinated team-roles, TR6 and TR7. In order to do that, TR5 has to communicate with TR6 and TR7 acting himself as the coordinator, and TR6 and TR7 as the operator (each one in a different scene).

Moreover, we introduce another role that is a superclass of both the coordinator and the operator roles, the *Problem-Solving Agent* (PSA) role.

$$(PSA \succeq Coordinator) \wedge (PSA \succeq coordinator) \tag{5.1}$$

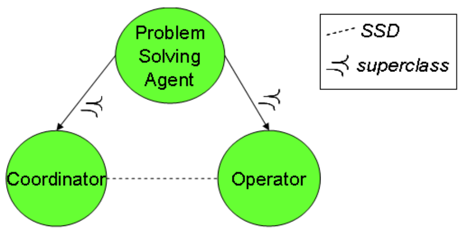


Figure 5.8: Basic roles and role relationships

Consequently, any agent enabled playing a PSA role can play also the coordinator and the operator roles (albeit never in the same scene instance). Figure 5.8 depicts the relationships among the basic ORCAS roles. The PSA role is a superclass of both the coordinator and the operator roles, and there is a static separation of duties between the coordinator and the operator roles.

Figure 5.9 summarizes the specification of the dialogic framework in the ORCAS ACDL: A shared Teamwork ontology, and the coordinator and operators as the only roles. There is a static separation of duties (SSD) between the coordinator and the operator roles. This relation means that an agent cannot be coordinator and operator within the same scene, since it has no sense for an agent to communicate with himself. There are no internal roles and both the content language and the illocutionary particles remain open (the example shows the typical illocutions and specifies NOOS as the content-language).

ontology	***Teamwork-ontology***
illocutionary-particles	*e.g. (request inform agree refuse inform failure)*
internal-roles	$\emptyset$
external-roles	*(**coordinator operator**)*
social-structure	*(**coordinator SSD operator**)*
content-language	e.g.*NOOS*

Figure 5.9: Dialogic frameworks in the ORCAS ACDL

Scenes

A scene is the main element used to describe the communication features of a capability, it describes what is commonly known as an agent interaction protocol. The same capability can support different interaction protocols, and thus a scene is required to specify each interaction protocol.

Recall that a scene is a conversation protocol shared by a group of agents playing some roles. More precisely, a scene defines a generic pattern of conversation among roles. Any agent participating in a scene has to play one of its roles. A scene is generic in the sense that it can be repeatedly played by different groups of agents, in the same sense that the same theater scene can be performed by different actors playing the same roles.

Electronic institutions use finite state machines (FSM) to specify scenes[1], which are represented by finite, directed graphs. A scene is defined as follows in the electronic institutions formalism [Esteva, 1997]:

Definition 5.6 ***(Scene)*** *A scene is a tuple:*

$$s = \langle R, DF, W, w_0, W_f, (WA_r)_{r\in R}, (WE_r)_{r\in R}, \Theta, \lambda, min, Max\rangle$$

where

- *R is the set of roles of the scene;*
- *DF is a dialogic framework (Definition 5.5);*
- *W is a finite, non-empty set of scene states;*
- *$w_0 \in W$ is the initial state;*
- *$W_f \subseteq W$ is the non-empty set of final states;*
- *$(WA_r)_{r\in R} \subseteq W$ is a family of non-empty sets such that WA_r stands for the set of access states for the role $r \in R$;*
- *$(WE_r)_{r\in R} \subseteq W$ is a family of non-empty sets such that WE_r stands for the set of exit states for the role $r \in R$;*
- *$\Theta \subseteq W \times W$ is a set of directed edges;*
- *$\lambda : \Theta \longrightarrow L$ is a labelling function, where L can be a timeout, an illocution scheme or an illocutions scheme and a list of constraints;*
- *min, Max are two functions that return respectively the minimum and maximum number of agents that can play a role $r \in R$;*

The nodes of the scene graph represent the different *states* (W) of the conversation, and the directed *edges* (Θ) connecting the nodes are labelled (λ) with the actions that make the scene state evolve: illocution schemes and timeouts. The graph has a single *initial state* (w_0, non-reachable once left) and a set of *final states* (W_f) representing the different endings of the conversation (there is no edge connecting a final state to another state).

A scene allows agents either to join it or leave it at specific states during an ongoing conversation, depending on their role. For this purpose, the sets of *access* (WA_r) states and *exit* states (WE_r) are differentiated for each role.

Normally the correct evolution of a conversation protocol requires a certain number of agents for each role involved in the scene. Thus, a *minimum* (*min*)

[1]An account of the reasons to adopt such a formalism is found in [Esteva, 1997], while another examples on using FSMs to specify agent conversations can be found in [Barbuceanu and Fox, 1995, Nodine and Unruh, 1999, d'Inverno et al., 1998]

and a *maximum* (Max) number of agents per role is defined and the number of agents playing each role has to be always between them.

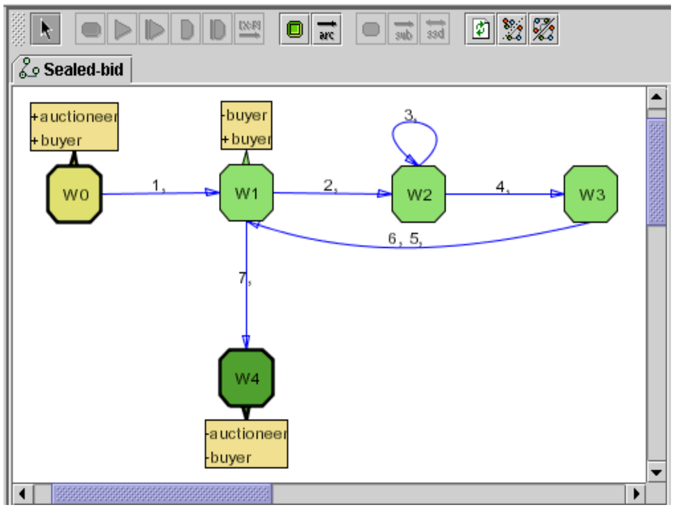
The final states have to be an exit state for each role, in order to allow all the agents to leave when the scene is finished. On the other hand, the initial state has to be an access state for the roles whose minimum is greater than zero, in order to start the scene.

The information exchanged between agents is expressed in the form of illocution schemes from the scene dialogic framework. In order for the protocol to be generic some details have to be abstracted. This means that state transitions cannot be labelled by grounded illocutions. Instead, illocutions schemes must be used, where the terms referring to agents and time are variables while the other terms can be variables or constants.

The other element that can label an edge is a timeout. Timeouts trigger on transitions after a given number of time units have passed since the state was reached. This is specially important for robustness —to evolve from states where agents dying and hence not talking any more, or where agents trying to foot-drag the other agents by remaining silent, could block the scene execution.

In addition to defining the valid sequences of illocutions that agents can exchange, a scene establishes the conversation context. Context is a fundamental aspect that humans use in order to interpret the information they receive. The same message in different contexts may certainly have a different meaning. Thus, a scene establishes what can be said, by who, and to whom, and allows to specify how past interactions may affect the future evolution of the conversation. The contextual information may restrict the valid messages in a certain state of the conversation. For instance, imagine a scene auctioning goods following the English auction protocol: as bids are submitted by buyers the valid bids for them are reduced to bids greater than the last one. That is to say, each submitted bid reduces the valid illocutions that buyers can utter, although the scene may continue in the same state. Such contextual information is encoded as constrains, which are used to restrict the set of values to create new bindings of the variables in the illocution schemes, as well as the paths that a scene conversation can follow. The reader is referred to [Esteva, 1997] for a detailed account on the use of variables and constrains in the electronic institutions formalism.

A scene has both a textual and a graphical representation. Figure 5.10 shows an example of an auction scene specifying a sealed-bid protocol. In this scene the participating agents can play the auctioneer and buyer roles. The graph depicts the states of the scene, along with the edges representing the legal transitions between scene states which are labelled either with illocution schemes of the communication language (according to elements defined within a dialogic framework) or with timeouts. Notice that apart from the initial and final states, the $w1$ state is labelled as an access and exit state for buyers, which means that buyers can leave and new buyers might be admitted between bidding rounds. Variable identifiers appearing in the illocution schemes can start with either '?'



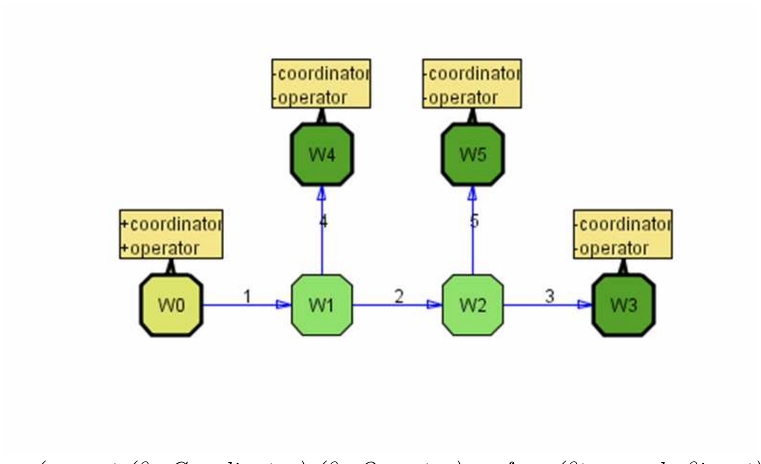
1 $(inform\ (?x\ auctioneer)\ (buyer)\ open\_auction(?r))$
2 $(inform\ (!x\ auctioneer)\ (buyer)\ start\_round(?good\_id, ?bidding\_time,$ $?reserve\_price))$
3 $(commit\ (?y\ buyer)\ (!x\ auctioneer)\ bid(!good\_id, ?offer))$
4 $!bidding\_time$
5 $(inform\ (!x\ auctioneer)\ (buyer)\ sold(!good\_id, ?price, ?winner))$
6 $(inform\ (!x\ auctioneer)\ (buyer)\ withdrawn(!good\_id))$
7 $(inform\ (!x\ auctioneer)\ (buyer)\ end\_auction(!r))$

Figure 5.10: Specification of a sealed-bid auction protocol

or '!', which is used to differentiate whether the variable can be bound to a new value or must be substituted by its last bound value.

Using scenes in ORCAS

In the electronic institutions formalism, when a new institution is defined, there is a global view of the system and thus it is possible to define in advance all the roles that can be played by the participating agents. Scenes in ORCAS are used to describe the patterns of interaction required to delegate a task to other agent, exclusively from the point of view of the agent providing the capability required to achieve that task. An agent providing a capability does not know in advance the potential team-roles it can get to play using that capability. Consequently, a scene describing the communication requirements of a capability can not be specified in terms of team-roles. The ORCAS approach to deal with issue is to define scenes in terms of two generic roles: the *coordinator* and the *operator* roles. The agent applying a task-decomposer and willing to delegate some subtask takes the coordinator role to communicate with each of the agents assigned to a subtask within the task-decomposer, engaging in a new scene for each task being delegated. Realize that the communication between an agent applying a task-decomposer and an agent responsible of one subtask is necessary only both the agents are the same; otherwise, if the agent assigned a task-decomposer were the same assigned a subtask, then there is no need for communication for that subtask.



1 *(request (?x Coordinator) (?y Operator) perform(?team-role ?input))*
2 *(agree (!y Operator) (!x Coordinator) perform(!team-role !input))*
3 *(inform (!y Operator) (!x Coordinator) result(!team-role ?ouput))*
4 *(refuse (!y Operator) (Coordinator null) perform(!team-role !input))*
5 *(failure (!x Operator) (!y Coordinator) perform(!team-role !input))*

Figure 5.11: Request-Inform protocol described by a scene

Figure 5.13 shows a scene specifying a FIPA-like Request-Inform protocol for a capability C. There are two roles, the *coordinator* and the *operator* roles. The coordinator role is played by the agent that has to delegate a task T to the agent providing C (assuming that T can be solved by C). The operator role is played by the agent providing C. The coordinator is the initiator of the scene, that begins with the coordinator sending a "request" message to the operator. That message contains the identifier of the team-role to be played by the operator, and the data to be used as input. The operator checks whether it is assigned to that *team-role*, and sends either an "agree" or a "refuse" message accordingly (if an agent is assigned to a team-role, it is supposed to agree, owing to the commitment implicit on an agent accepting a team-role during the Team Formation process). The operator agent holds the information required to carry out its accepted team-roles, as that information was provided during the Team formation process; therefore, the operator can solve the task assigned to that team-role by applying the capability bound to it, using the data provided as input by the coordinator, and the selected domain knowledge. If the capability is applied to the input data successfully, then the operator sends the result to the coordinator with an "inform" message, otherwise the operator sends a "failure" message. There are three final states that are reached when the operator refuses the request from the coordinator ($w4$), the application of the capability fails ($w5$), or it ends successfully ($w3$).

A wide range of communication styles can be specified using scenes, from very simple protocols involving two roles to very complex interaction protocols with several agent roles, time-outs, transition constrains and so on, thus giving quite expressiveness to developers. Nevertheless, using a reduced set of standardized basic protocols is encouraged in ORCAS to maximize capability reuse. In order to have an intuitive idea of possible styles of interaction we can consider the basic interaction protocols defined for Web services, which are called operations in WSDL and processes in DAML-S [The DAML-S Consortium, 2001]. There are four basic types of "operations" according to these proposals:

- *request-response* operation (an atomic process with both inputs and outputs in DAML-S);
- *one-way* operation (an atomic process with inputs but no outputs);
- *notification* operation (an atomic process with outputs, but no inputs);
- *solicit-response* operation (a composite process with both outputs and inputs, and with the sending of outputs specified as coming before the reception of inputs).

The *request-response* operation corresponds to the request-inform protocol showed in Figure 5.13 as an example of a scene. The *one-way* and the *notification* operations can be specified by the same scene but with different edge labels, as showed in Figure 5.12. Finally, the solicit-response operation requires some minor changes in the scene; there is required a new state, w_6, and a new transition (Transition 6) from $w5$ with an inform message to be send by the Coordinator to the Operator, containing the input.

	One-way operation (only input)
1	*(request (?x Coordinator) (?y Operator) perform(?team-role ?input))*
2	*(agree (!y Operator) (!x Coordinator) perform(!team-role !input))*
3	*(inform (!y Operator) (!x Coordinator) done(perform(!team-role !input))*
4	*(refuse (!y Operator) (Coordinator null) refusal(perform(!team-role !input) reason)*
5	*(failure (!x Operator) (!y Coordinator) failure(perform(!team-role !input) reason)*

	Notification: (only output)
1	*(request (?x Coordinator) (?y Operator) perform(?team-role))*
2	*(agree (!y Operator) (!x Coordinator) perform(!team-role))*
3	*(inform (!y Operator) (!x Coordinator) result(!team-role !output))*
4	*(refuse (!y Operator) (!xCoordinator) perform(!team-role))*
5	*(failure (!x Operator) (!y Coordinator) perform(!team-role))*

	Solicit-Response (both input and output, but input cames after the output is provided
1	*(request (?x Coordinator) (?y Operator) perform(?team-role))*
2	*(agree (!y Operator) (!x Coordinator) perform(!team-role))*
3	*(inform (!y Operator) (!x Coordinator) result(!team-role !output))*
4	*(inform (!x Coordinator) (!x Operator) response(!team-role !input))*
5	*(refuse (!y Operator) (Coordinator null) inform(!team-role ?input))*
6	*(failure (!x Operator) (!y Coordinator) perform(!team-role))*

Figure 5.12: Variations and alternatives to the Request-Inform protocol

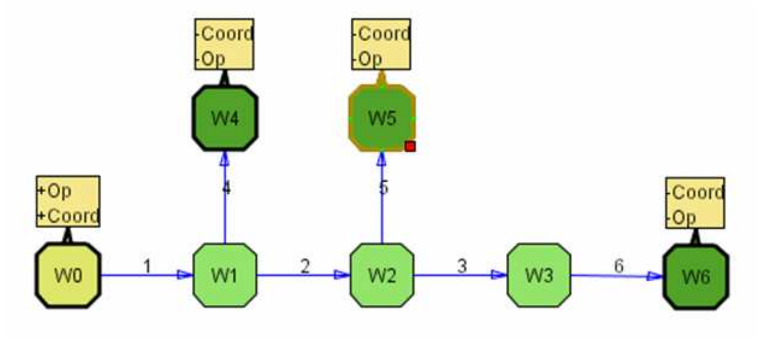


Figure 5.13: Solicit-Response protocol described by a scene

Summing-up, scenes provide a flexible and precise (not ambiguous) way of defining interaction protocols to communicate with agents in order to delegate tasks to other agents. Every time an agent has to delegate a task to another in the context of a teamwork process, it has to engage in communication with the agent assigned to the task's team-role. This communication will follow the specification of a scene supported by the agent providing that capability. The decision on which specific scene is to be used between two agents is negotiated during the Team Formation process, as explained in §5.5.

5.4.3 Operational description

The purpose of the *operational description* is to specify the data and control flow among the subtasks of a task-decomposer. Control flow languages are based on the notion of control constructs like iteration, parallelism, branching conditions and so on. Control flow modelling has been a research area that attracted significant interest in the last decade. Nevertheless, little consensus has been reached as to what the essential ingredients of a control flow specification language should be, and there are notable differences in the expressive power of control flow specification languages [Kiepuszewski, 2002].

Since our approach is based on Knowledge Modelling frameworks we have considered some proposals from the Knowledge Modelling community, like KARL [Fensel et al., 1998a] and Modal Change Logic [Fensel et al., 1998b], but we found that these proposals rely on a sequentiality assumption that is not appropriate for MAS. Therefore, it seems more appropriate to use agent concepts for describing the interaction among subtasks in order to deal with parallelism. Such a language must capture dependency relationships, temporal relationships, and parallelism in order to support the team configuration during the Team Formation process, and the coordination of team mates during the Teamwork process. In order to describe these aspects, we will continue using the concepts on electronic institutions: specifically, we will use the notion of a performative structure to describe the operational description of task-decomposers.

We want to increase the reusability of capabilities by describing the operational description of task-decomposers from a compositional approach, maximizing the reuse of capabilities by keeping them separated from both the tasks and the domain models. In addition, we want to use a formalism supporting parallelism and allowing for the specification of synchronization points, and multiple task instantiation (i.e. tasks that can be played several times during the same teamwork process).

In order to deal with these issues, the ORCAS ACDL proposes the specification of the operational description as a task network, using the concept of a *performative structure* from the electronic institutions formalism. The point is to describe the operational description of a task-decomposer as a composition of several scenes, where each scene corresponds to a scene describing the communication between the agents playing the coordinator and the operator roles for that capability during the Teamwork process.

While a scene models a particular multi-agent dialogical activity, more complex activities can be specified by establishing relationships among scenes. This issue arises when conversations are embedded in a broader context, such as, for instance, organizations and institutions. If this is the case, it does make sense to capture the relationships among scenes. For these purpose a performative structure defines which are the scenes of the electronic institution and the *role flow policy* among them. That is to say, *how* the agents can move among the different scenes depending on their role, and *when* new scenes have to be started, taking into account the relationships among the different scenes.

Performative structure

A performative structure is a network of connected scenes that captures the relationships among scenes. The specification of a performative structure contains a description of how the different agent roles can move from one scene to another. More formally, a performative structure is defined as follows [Esteva, 1997]:

Definition 5.7 ***(Performative Structure)*** *A performative structure is a tuple*

$$PS = \langle S, T, s_0, s_\Omega, E, f_L, f_T, f_E^O, C, \mu \rangle$$

where

- *S is a finite, non-empty set of typed scenes, where each scene is defined by a name (S_{name}) and a type (S_{type}) (Def. 5.6);*
- *T is a finite and non-empty set of transitions;*
- *$s_0 \in S$ is the* initial *scene;*
- *$s_\Omega \in S$ is the* final *scene;*
- *$E = E^I \bigcup E^O$ is a set of edge identifiers where $E^I \subseteq S \times T$ is a set of edges from scenes to transitions and $E^O \subseteq T \times S$ is a set of edges from transitions to scenes;*
- *$f_L : E \longrightarrow V$ maps each edge to an edge label V, represented as a disjunctive normal form (DNF)[2] in which literals are pairs composed of an agent variable and a role identifier representing an edge label;*
- *$f_T : T \longrightarrow \mathcal{T}$ maps each transition to its type;*
- *$f_E^O : E^O \longrightarrow \mathcal{E}$ maps each edge to its type;*
- *$C : E^I \longrightarrow CONS$ maps each edge to a expression representing the edge's constraints.*

[2]Disjunctive Normal Form or DNF is a method of standardizing and normalizing logical formulae. A logical formula is considered to be in DNF if and only if it is a single disjunction of conjunctions. More simply stated, the outermost operators of the formula are all ORs, and there is only one level of nesting allowed, which may only contain literals or conjunctions of literals

- $\mu : S \longrightarrow \{0, 1\}$ *establishes whether a scene can be multiply instantiated at execution time;*

A performative structure contains a set of typed *scenes* (S). The scene type is the specification of the scene according to Definition 5.6), thus different scenes within a performative structure can refer to the same type of scene. There are two scenes defined as the initial (s_0) and final (s_Ω) scenes. Relationships among scenes are specified as *transitions* (T) agents must traverse in order to move from one scene to another, and edges going from scenes to transitions (incoming edges, E^I), and from transitions to scenes (outgoing edges, E^O). In order to move from one scene to another, an agent has to progress through a *transition* (direct connections between scenes are forbidden). In general, the activity represented by a performative structure can be depicted as a collection of multiple, concurrent scenes, and agents navigating from scene to scene constrained by transitions. A performative structure can specify also whether a scene can be multiple instantiated or not at execution time (μ).

The edges of a performative structure are labelled so as to specify which agents can progress through an edge depending on their roles. These labels are expressed as conjunctions and disjunctions of pairs composed of an agent variable and a role identifier. The role identifier determines which type of agent is allowed to follow the edge, while agent variables are used to differentiate among agents playing the same role. For instance, an edge labelled with $(x\,R_1) \wedge (y\,R_2)$ means that this edge can be followed only by pairs of agents where one of them is playing the role R_1 and the other is playing the role R_2. On the other hand, an edge labelled with $(x\,R_1) \vee (y\,R_2)$ means that any agent playing either the role R_1 or the role R_2 can progress through that edge. The scope of an agent variable includes all the incoming and outgoing edges of a transition. That is to say, if an agent reaches a transition following an incoming edge labelled with $(x\,R_1)$, it can only leave the transition by following those outgoing edges containing the variable x in their label. However, there is a relation between the agent variables labelling the incoming and the outgoing edges of a scene. A conjunction over a incoming edge (from a scene to a transition) means that the agents have to leave the scene together (and reach the transition together too), whereas a conjunction labelling an outgoing edge (from a transition to a scene) means that the agents must enter the target scene together, that is to say, agents must enter into the same scene instance.

There are two types of transitions (f_T) according to how agents coming from several edges can progress through them:

- **AND** transitions establish synchronization points and parallelism. Agents reaching a transition from several incoming edges have to wait for agents coming from all the incoming edges in order to progress through the transition, and must follow all the outgoing edges where they appear (the variables they are bound to).
- **OR** transitions allow agents to progress through them in an asynchronous way and are used to define choice points. Agents reaching an OR transition

can progress through it without waiting for other agents, and are allowed to choose which outgoing edge to follow when leaving the transition.

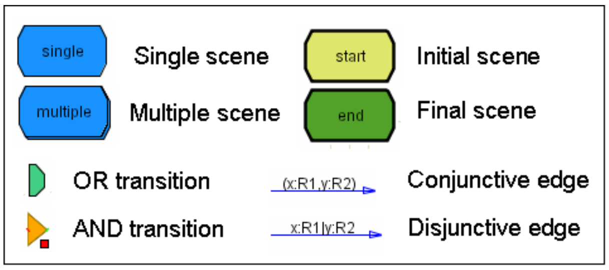


Figure 5.14: Graphical elements used to specify a performative structure

Figure 5.14 shows the graphical elements used to specify a performative structure: scenes, transitions and edges. There are different symbols to distinguish the initial and final scenes from other scenes, and to differentiate whether a scene can be multiple instantiated or not (single). Notice there are two types of transitions, AND and OR; and two ways of labelling an edge, conjuntion and disjuntion.

An example of a performative structure for an agoric market is shown in Figure 5.15 (extracted from [Esteva, 1997]). The root scene is the Admission scene, where any agent enters the institution. Buyers and sellers can move from the Admission scene to the Agora scene, where they can try to buy and sell goods. When a buy/sell operation is agreed, both the involved buyer and seller together meet with an accountant agent in the Settlement scene to formalize the operation. Finally, agents can exit the institution by reaching the Departure scene.

Using performative structures in ORCAS

Our approach to specify the operational description of a task-decomposer is based on performative structures, with some distinctive features.

A first feature characterizing the use of performative structures to specify the operational description of a task-decomposer arises from the fact that the precise team-role applying a task-decomposer is not known, because a task-decomposer is defined for a capability, independently of any particular task that can solved applying that capability. Our approach to overcome this problem is the use of two generic roles, as explained in §5.4.2 and §5.4.2: the coordinator and the operator roles. Therefore, when defining an operational description as a performative structure, we know that the agent providing a task-decomposer adopts the role of the coordinator, while the agents assigned to every subtask adopt the operator role, and thus, the performative structure can be defined in terms of these generic roles.

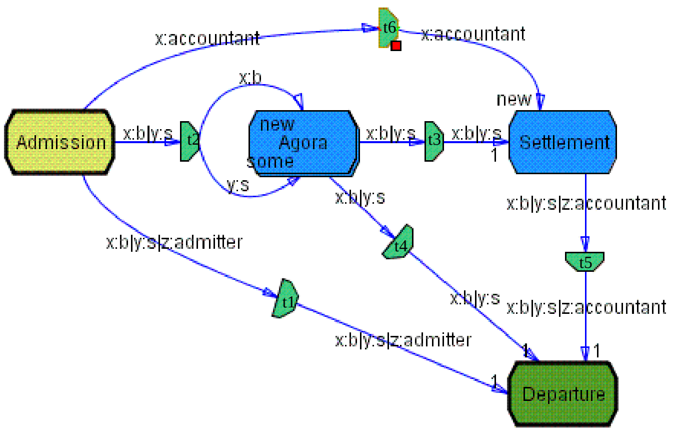


Figure 5.15: Performative structure of an agora

The second feature distinguishing a performative structure in ORCAS from the electronic institutions approach results from the decoupling of tasks and capabilities. In ORCAS, each scene within the performative structure corresponds to an interaction protocol describing the communication required between a task-decomposer team-role acting as a coordinator, and a subordinated team-role acting as operator. In other words, each scene within the performative structure refers to a task to be delegated to another agent. In ORCAS, each task (T) is solved using a capability (C); therefore the coordinator agent has to communicate with the agent providing C, using one of the scenes supported by that agent (acting as operator) over C. Consequently, the specific scenes to be used within a performative structure must be decided before starting the Teamwork process. In ORCAS this process is carried over during the Team Formation process. The goal is to select the scenes from those supported from both the provider and the requester of a capability. We note A^C an agent providing C and A^C_S as the set of scenes it supports over C. The provider of a capability must play the operator role and the requester must play the coordinator role. Both the coordinator and the operator must follow the same scene in order to communicate, and as a consequence, the scene must be chosen out of the scenes supported by both agents (the intersection of two set of scenes, the scenes supported by the coordinator, and the scenes supported by the operator).

In order to specify the operational description of a task-decomposer, we adapt the notion of a performative structure to fit better in the ORCAS framework. Specifically, the constraints and the edge typing functions are not used in an operational description, the scenes are not typed ($\dashv$), and two scenes called

Start and End are defined as the initial and final scenes. More formally, an operational description in ORCAS is defined as follows [Esteva, 1997]:

Definition 5.8 ***(Operational description)*** *A operational description is defined for a task-decomposer D as a tuple*

$$OD(D) = \langle S, T, s_0, s_\Omega, E, f_L, f_T, f_E^O, \mu \rangle$$

where

- *S is a set of untyped scenes named after the subtasks introduced by the task-decomposer, plus an initial and a final scene called Start and End respectively;*
- *$s_0 \in S =$ Start (the initial scene);*
- *$s_\Omega =$ End (the final scene);*
- *T is a finite and non-empty set of transitions;*
- *$E = E^I \bigcup E^O$ is a set of edge identifiers where $E^I \subseteq S \times T$ is a set of edges from scenes to transitions and $E^O \subseteq T \times S$ is a set of edges from transitions to scenes;*
- *$f_L : E \longrightarrow V$ maps each edge to an edge label V, represented as a disjunctive normal form (DNF) over pairs composed of an agent variable and a role identifier representing an edge label;*
- *$f_T : T \longrightarrow \{AND, OR\}$ maps each transition to its type;*
- *$\mu : S \longrightarrow \{0, 1\}$ establishes whether a scene can be multiply instantiated at execution time;*

Notice that the set of scenes of an operational description has no type, that is to say, scenes are not bound to a scene specification, but they have just a name. Moreover, scenes are named as the subtasks of the task-decomposer, plus two scenes called Start and End.

Figure 5.16 shows an example of a performative structure specifying the operational description of the Aggregation task-decomposer, which decomposes a task into two subtasks: Elaborate-items and Aggregate-items. Therefore, the performative structure has two scenes (in addition to the Start and End scenes), one for each subtask. There are three roles involved in the performative structure, a coordinator to be played by the agent applying the task-decomposer, and as many operators as subtasks. In the example there are two operators, one (y) participating in the Elaborate-items (EI) scene, and another (z) participating in the Aggregate-Items (AI) scene. Notice that the coordinator (x) should be the same in both scenes, it enters first the EI scene, and can enter the AI scene only after finishing the EI scene.

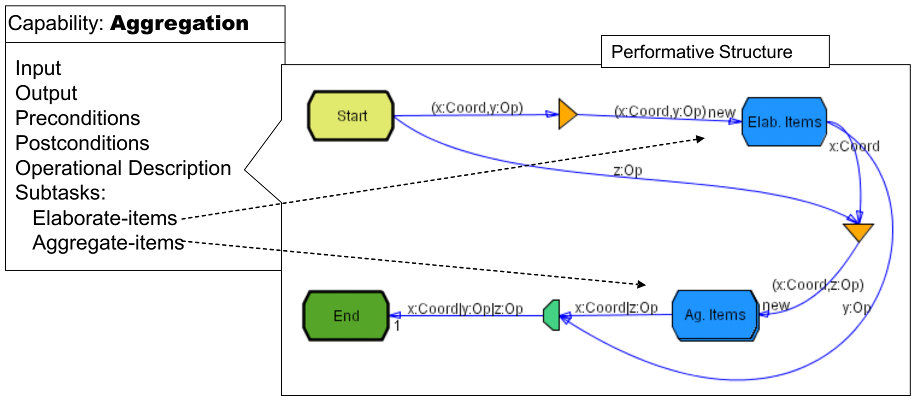


Figure 5.16: Task-decomposer operational description

The task-based performative structure used for specifying the operational description of a task decomposer keeps the decoupling of tasks and capabilities. This approach, which aims at maximizing capability reuse, leads to the use of scenes based on two generic roles. And the low granularity of the two-role scenes used as the building blocks of a performative structure, together with the existence of the-facto standards for one-to-one interaction (e.g. the FIPA Request-Inform protocol), are two extra features supporting the goal of maximizing reuse.

Using performative structures to describe the operational description of a capability enables parallelism and provides an abstract view of the coordination required for Teamwork, which can be sensibly used to improve the Team Formation process by producing more robust teams and improving the overall performance of the team.

We have explained the ORCAS model of the Cooperative Problem Solving process, and the ORCAS team-model. Moreover we have introduced a formalism to describe the communication and the operational aspects required to turn the Knowledge Modelling Ontology into a full-fledged ACDL. Now we are in position to focus on the two operational stages of the CPS process: Team Formation and Teamwork.

5.5 Team Formation

Team Formation is the process of selecting a group of agents that have complimentary skills to achieve a global goal (the team goal), and providing team members with the information required to achieve the global goal in a cooperative way.

Team Formation in ORCAS is guided by a task configuration. Since a task-configuration specifies the tasks to be solved and the capabilities to apply, the number of possible teams is reduced, making Team Formation feasible in practice. In large systems, team selection may involve an exponential number of possible team combinations, and a blow-out in the number of interactions required to select the members of a team [Kinny et al., 1992]. ORCAS addresses this problem by introducing the Knowledge Configuration process before Team Formation in the Cooperative Problem-Solving process [Wooldridge and Jennings, 1999].

Our model of the Team Formation process considers three activities, namely: *task allocation*, *team selection* and *team instruction*.

- During the *task allocation* process candidate agents are obtained for each task, according to the requirements of a task-configuration;

- next, during the *team selection* process, some agents are selected for each specific team-role, while other agents are kept in reserve for the case of agent failure;

- finally, during the *team instruction* process, agents involved in the team formation process are informed about the result of the team configuration stage: the team roles they have to play, and the social knowledge required to cooperate with other team-members during the Teamwork process.

Later, in Chapter 6 a particular agent infrastructure supporting the ORCAS framework is presented. This infrastructure provides the services required from both providers and requesters of capabilities to form customized teams of agent on-demand. The approach there is to include institutional agents acting as middle-agents with the capabilities required to configure tasks and coordinate the Team Formation and the Teamwork processes. Since there are a lot of strategies and algorithms for team formation and agent coalition formation, we want to explain Team Formation from a more conceptual point of view, focusing on the inputs, the outputs and the requirements of the Team Formation process, rather than describing how the process is carried on in the ORCAS implemented infrastructure, addressed in Chapter 6.

5.5.1 Task allocation

Task allocation is the process of selecting a group of candidate agents to form a team, such that their aggregated competence satisfies the requirements of the problem at hand. This process follows the task decomposition structure defined by a task-configuration to know which are the tasks to be allocated, and looks for candidate agents that are suitable to solve each task.

Task-allocation can be performed by a middle agent, facilitated by it, or distributed among several agents. Since the ORCAS implemented infrastructure aims at minimizing agent requirements and facilitate teamwork, the ORCAS infrastructure relies upon middle agents to support both providers and requesters of capabilities during the CPS process. Specifically, the ORCAS infrastructure provides a kind of broker called Team-Broker, which is able to form teams on-demand, according to a task-configuration.

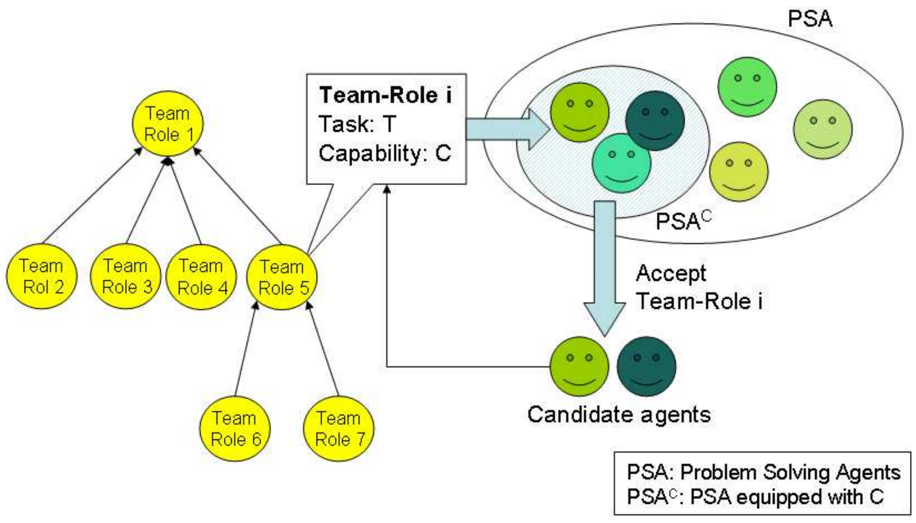


Figure 5.17: Task allocation

A team is defined as a hierarchy of team-roles derived from a task-configuration. Each subtask within a task configuration defines a team-role that should be played by someone, and more specifically, at least one agent must commit to each team-role in order to complete the task allocation process. Candidate agents are those than in addition to be equipped with the capability required for a team-role, accepts to play that team-role.

Figure 5.17 shows the task-allocation process as a filtering process. From the Problem-Solving Agents (PSA) available, only those equipped with a team-role's capability are potential candidate. At the end, only the agents accepting to play a team-role with the specified requirements (the capability and the domain-knowledge to use in order to solve the team-role's task) become candidate agents. The Task Allocation process proceeds until there are candidate agents for all the team-roles composing a team. In the example, there are three agents equipped with the required capability (C), which become potential candidates. In the end, only two agents have accepted to play that team-role.

Although we avoid establishing a model of commitment based on mental attitudes (joint intentions, joint commitment) some model of agency is required to implement the CPS process. Herein we will rely on a weak notion of agency, specifically, an implicit model of commitment will be assumed. From this approach, commitment is implicit in the act of accepting a team-role proposal; in other words, when an agent accepts a team-role proposal during the team selection process, one assumes that the agent is committing to achieve the corresponding task using the selected capability, as specified in the task-configuration.

In the ORCAS implementation of an agent infrastructure supporting the

Team Formation process, task allocation is a responsibility of the Team Broker role. Although individual agents may be self-interested, we assume that there is a global interest for the team to offer together a good service, and thus, some service provided by the infrastructure to select appropriate agents appears as an interesting feature, for instance, selecting the agents with lower workload, or agents with a cheaper cost, depending on the preferences specified for the problem at hand.

In the ORCAS infrastructure, a Team-Broker role is defined that is able to obtain candidate agents by using a task configuration as the source to generate team-roles, and sending team-role proposals to potential candidate agents. The agents receiving a team-role proposal can autonomously decide to agree, to refuse the proposal, or to make a counter-proposal. The Team-Broker role has to wait until all the available agents have answered or a time-out is reached. Candidate agents will be used to select the members of the team, as explained in the next subsection.

5.5.2 Team selection

Team selection is the process of selecting a set of team members from the collection of candidate agents obtained during the task-allocation process.

The result of the Team Selection process in ORCAS is a *team-configuration*, which results of instantiating the abstract team-roles of a team model with specific agents selected from candidate agents. A team-configuration is obtained by selecting a group of agents, and optionally some reserve agents, for each team-role. All agents selected and kept in reserve for some team-role become team-members.

A team-configuration is complete when all the team-roles composing a team (Definition 5.4) are complete:

$$Complete(Team(\pi^0, Conf(\kappa))) \iff \forall \pi \in Team(\pi^0, Conf(\kappa)) : \pi_{A_S} \neq \emptyset$$

where π_{A_S} is the set of agents selected to play team-role π. Otherwise $Team(\pi, Conf(\kappa))$ is partial. In other words, a team-configuration is complete when there are agents selected to play all the team-roles composing a team.

Figure 5.18 shows an example of a team-configuration. In particular, this example shows a team-configuration for a team that has to solve the Information-search task. Some team-roles are assigned a single agent (TR1, TR2, TR5, TR6), while other team-roles have several agents selected (TR3, TR4 and TR7), besides some team-roles include reserve agents (TR1, TR4). In some cases, the agent selected to apply a task-decomposer has to delegate all the subtasks to other agents (TR1), while in other cases the task-decomposer agent is assigned some (or all) of their own subtasks (TR5).

A second goal of team selection is to decide the scenes to be used between agents requiring some communication. Since ORCAS teams are hierarchically organized, every interaction occurring during the Teamwork process involves an agent playing a task-decomposer team-role and acting as the coordinator, and

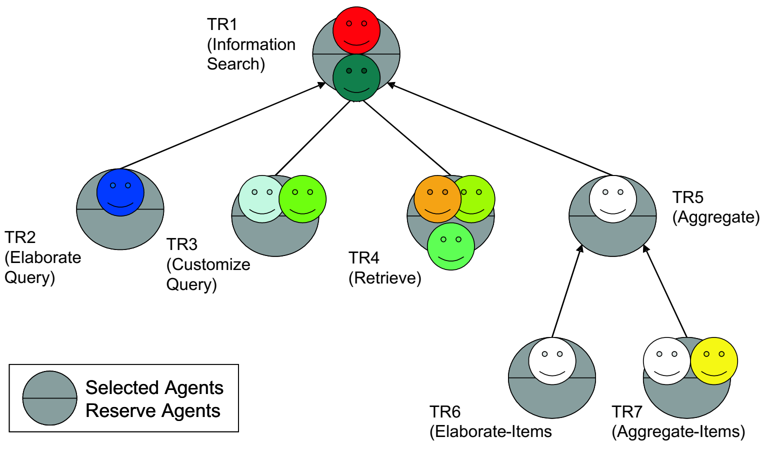


Figure 5.18: Example of Team-configuration

one agent playing a subordinated team-role (there is one subordinated team-role for each subtask). The coordinator is responsible for decomposing the problem and delegating the subtasks to the agents selected for a subordinated team-role, distributing data to other agents, receiving back the results, and performing intermediate data processing between subtasks.

Team selection in the ORCAS Operational Framework can use different selection criteria and can be carried on according to different strategies and interaction protocols. There are just a few requisites imposed by the ORCAS operational framework to be satisfied by the Team selection process:

- Selecting at least one agent for each team-role, except when there are alternative team-roles, since then, an agent selected for any of the alternative team-roles is enough.
- Selecting a scene for each team-role, such that the scene is shared by both the agent willing to play the coordinator team-role, and the agent willing to play the operator team-role. In other words, two agents must share a common scene in order to communicate.
- Optionally, non-selected candidate agents can be kept as reserve agents for the case the selected ones could not achieve their task.

In the ORCAS infrastructure, described in Chapter 6, the team selection process is performed through an auction-like protocol driven by the Team-Broker role.

Figure 5.19 sums up the process of choosing the communication scenes to be used for each team-role. First of all, one or more agents are selected to play every team-role. Secondly, a single scene is selected for every team-role. These scenes are selected from the scenes shared by both the agent willing to act as the operator, and the agent willing to act as the coordinator. In the example, agent

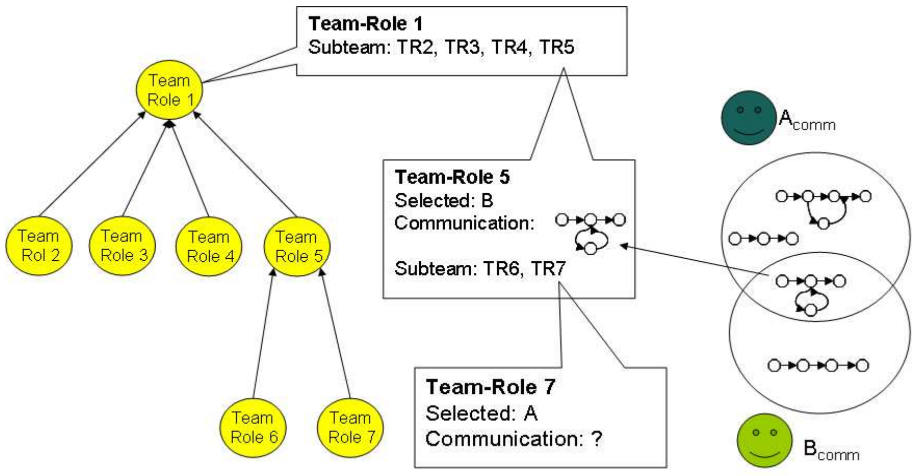


Figure 5.19: Choosing communication scenes during the team selection process

B is selected to play the operator role (Team-Role 7), and agent A is selected to play the coordinator role (Team-Role 5). The scene specifying the communication between both roles is selected from the intersection of the communication scenes supported by both agents, and is assigned to the communication slot of Team-role 1 (the one playing the operator role).

Summarizing, the result of the Team selection process is a *team-configuration*, a hierarchical structure of interrelated team-roles complying with a task-configuration. Each team-role within a team-configuration defines the following elements (Definition 5.1): a task to be achieved; a capability to achieve the task; a set of agents selected to play the team-role; a set of agents to keep in reserve; a communication scene specifying the interaction protocol; optionally, the domain knowledge to be used by the capability; and finally, if the capability is a task-decomposer, a team-role must include a subteam feature describing the team-roles subordinated to this team-role (see §5.3.1), and specified as team-components (Definition 5.2).

Although there are multiple strategies allowed by the ORCAS framework to assign agents to team-roles (to allocate tasks to agents), there are some general considerations to take into account.

On the one hand, agents can play more than one role in a team and thus they can be selected to occupy several positions. Consequently, an agent playing both a task-decomposer team-role and some of the subordinated team-roles can reduce communication costs by performing the tasks assigned to both the task-decomposer and the subordinated roles. However, agents may be selected taking into account their workload, in order to balance the global performance of the MAS, that can be performing multiple tasks in parallel.

On the other hand, the ORCAS approach aims at exploiting the information

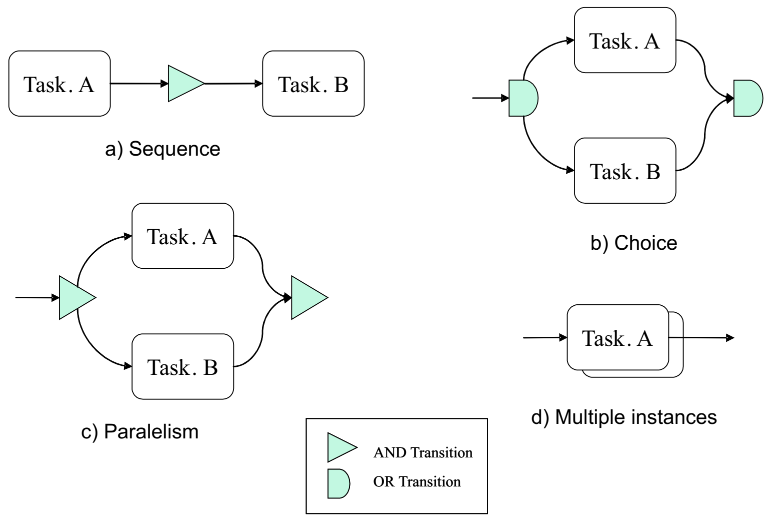


Figure 5.20: Representing control-flow in a performative structure

provided by the operational description of a task-decomposer to know which are the task dependencies. This information can be used when selecting the agents that will play each team-role, trying to increase the possibility of success and improving the overall team performance. We consider four types of task relationships that can be very useful during the Team Formation process, namely sequences, choices, parallelism and multiple-instances.

In order to characterize the task relationships involving a group of alternative (choices) or parallel tasks, we will use the term “fade-in” transition to refer to an initial transition agents are forced to traverse before performing any of the tasks in the group, and the term “fade-out”is used to denote the transition agents must reach in the end, after performing any of the tasks. The fade-in transition has outgoing edges going to the alternative/parallel tasks, and the fade-out transition has incoming edges coming from all these tasks. We describe below the four types of relationships considered within the ORCAS framework, and discuss briefly the way they can be used to improve the Team Formation process:

- *Sequences* (Figure 5.20.a) are defined among tasks than should be solved one after another. Usually, two tasks should be performed sequentially when there is some data dependencies between them (the output of one task is the input of another task). Tasks to be performed sequentially have an AND transition between them. Sequential tasks do not allow parallelism, therefore it is not advantageous to select different agents to solve them. The agent selection criteria for sequential tasks is independent from the other tasks.

- *Choices* (Figure 5.20.b) are used to define alternative tasks to choose from.

An agent faced with a set of alternative tasks can choose any of them to progress through the performative structure. Choices are specified by a set of alternative tasks preceded by an OR (fade-in) transition and followed by an OR (fade-out) transition too. There are an outgoing edge from the fade-in OR transition to each alternative task, thus an agent traversing that transition can move to any of the following tasks by choosing one among the outgoing edges. The OR transition after the alternative tasks allow an agent having performed some task to progress through it without waiting for agents performing other tasks. In general, one single agent is sufficient to allocate a set of alternative tasks, since only one task is strictly to proceed further. However, some times may be useful to try several alternative tasks and then retain the result of only one task, for instance, the first finished task. Therefore, when there are candidate agents for several alternative tasks, it might be interesting (though not necessary) to select agents suitable for several alternative tasks.

- *Parallelism* (Figure 5.20.b) means that several tasks must be performed in parallel. Parallel tasks are represented between an AND fade-in transition, an AND fade-out transition, and several outgoing edges connecting the fade-in transition to every task. All the tasks between the fade-in and fade-out must be performed in parallel. The fade-in (AND) forces agents traversing it to follow all the outgoing edges. The fade-out transition (AND) ensures that all the tasks to be performed in parallel have finished to allow agents to proceed further. In order to exploit parallelism the Team-Formation process should select different agents to perform parallel tasks.

- *Multiple instances* (Figure 5.20.b) means that a task can be performed multiple times in parallel. Multiple instances are represented by overlapped tasks within an ORCAS performative structure. For example, the Retrieve task appears within the Meta-search task-decomposer's operational description as allowing multiple-instances. This is due to the fact that Retrieve takes a single query as input, while the previous task (Customize-Query) produces a set of queries. For this reason the Retrieve task must be repeated for each query produced by the Customize-query task. Since multiple instances of a task may be solved in parallel, it could be beneficial to assign several agents to that task.

5.5.3 Team instruction

Team Instruction is the process of informing each team member about all the information they need to play their team-roles during the Teamwork process. Team-roles have been defined in Section §5.3, and we have stated that team-role structures are used during the Team Formation process not only to send proposals to join a team during the task allocation process, but also to instruct team members on the tasks they have to solve, the capabilities to apply, and all the information required to communicate with other team-members. Specifically, we use team-roles to inform an agent willing to apply a task-decomposer on which tasks to be delegated to other agents, to whom, and which communication scenes to use. A useful distinction is established distinguish between team-roles assigned to a skill, and team-role assigned to a task-decomposer.

Figure 5.21 shows an example of a team-role endowed with a skill. This team-role is associated to the task Elaborate-query, and specifies that the capability Query-elaboration-with-thesaurus, which is a skill, should be applied for solving Elaborate-query. Furthermore, that skill has to use the domain knowledge characterized by the MeSH thesaurus domain model.

Team-Role	
Task	Elaborate-Query
Team-Id	Team-23
Role-Id	Role-2
Capability	Query-elaboration-with-thesaurus
Domain-Models	MeSH-Thesaurus

Figure 5.21: Team-role example for a skill

Figure 5.22 shows an example of a team-role endowed with a task-decomposer. This team-role corresponds to the Information-search task, which is bound to the Meta-search capability. This capability is a task-decomposer introducing four subtasks: Elaborate-query, Customize-query, Retrieve and Aggregate. Therefore, the subteam for this capability has four team-components, one per subtask. Each team-component in the sub-team identifies both the agents selected to solve one of the subtasks, the agents to keep in reserve for each subtask, together with the task and the identifier of the team-role associated to that task. For instance, in Figure 5.22, the Red agent is selected to play the role assigned to the Elaborate-query task, while the Green agent is kept in reserve. However, sometimes it is desirable to allocate the same task to several agents. In that example there are two agents selected for the Retrieve task, Red and Blue, because that task may be executed multiple times while applying a meta-search capability: the task Retrieve takes a single query as input, while other tasks that are executed previously (Elaborate-query and Customize-query) usually output several queries; thus the task Retrieve has to be performed once for each query. Since these multiple executions may be carried over in parallel, it can be beneficial to distribute the multiple instances of the task among different agents.

The information provided by a team-role to a team-member agent during the team instruction process is used in the following way: an agent being requested to perform a task is provided with a team-identifier and a team-role identifier, so as to allow that agent to retrieve the information about the requested team-role from his local knowledge base. First of all, the agent checks whether is it committed to that team-role or not. Following, the agent finds out whether the

Team-Role				
Task	Information-Search			
Team-Id	Team-23			
Role-Id	Role-1			
Capability	Meta-Search			
Subteam				
	Subtask	*Role-Id*	*Selected*	*Reserve*
	Elaborate-quey	Role-2	Red	Green
	Customize-query	Role-3	Red	Yellow
	Retrieve	Role-4	Red, Blue	
	Aggregate	Role-5	Blue	Cyan

Figure 5.22: Team-role example for a task-decomposer

team-role's capability is a skill or a task-decomposer. If the team-role capability is a skill, then the agent applies that skill and sends the result back to the requesting agent. Otherwise the capability is a task-decomposer, and the agent has to check the team-role subteam to find out whether it has to delegate some subtask to another agent. This information is provided by team-components; if a team-component has selected agents different of himself, then the corresponding subtask must be delegated to that agent. The communication scenes to be used are also specified by the team-components.

To sum up, the team instruction process provides team-members with all the information they require activity to cooperate with other team-members during the Teamwork process.

5.6 The Teamwork process

The Teamwork process comprehends all the activities a team must carry out to solve a problem. In ORCAS the Teamwork process aims at solving a problem according to its requirements. In order to do that, during the Team Formation process a group of agents have joined a team by committing to some team-roles. The resulting team is customized for the problem at hand, since it is based on a task-configuration satisfying the stated problem requirements. Team members have been instructed on the tasks they have to solve (one task for each team-role), and on the capabilities they must apply to solve each task.

Whilst the Knowledge-Configuration process operates over static information describing agent capabilities from an abstract view, the Teamwork process has to take into account the dynamic nature of the environment. A team solving a problem in a real environment has to deal with events and conditions occurring at runtime, which may difficult the achievement of the team goals, e.g. excessive workload, agent failures or communication problems.

A team is composed of a task-coordinator playing a task-decomposer team-role, and a sub-team. The sub-team coordinator has to apply a task-decomposer capability to achieve its goals. The operational description of a task-decomposer describes the control flow over subtasks as a performative structure (§5.4.3).

The performative structure describing a task-decomposer's operational description is specified as a network of interrelated scenes, one for each subtask. Each scene requires at least two agents to be carried out: a *coordinator*, which is played by the agent assigned to the task-decomposer team-role, and one agent assigned to a subordinated team-role and playing the *operator* role. The formal scenes describing the communication required to achieve each task have been decided during the team selection process (§5.5.2). Consequently, in order to apply a task-decomposer, the coordinator agent has to initiate the different scenes while following the performative structure.

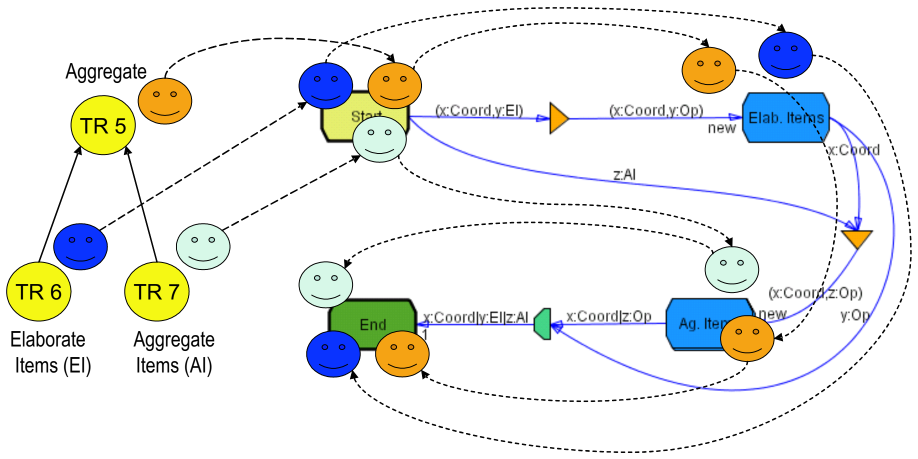


Figure 5.23: Teamwork model for a task-decomposer

Figure 5.23 shows the role flow policy through the performative structure describing the operational description of a task decomposer. Specifically, this figure shows the operational description of the Aggregation capability, which is bound to Team-role 5 (TR5), together with the paths to be followed by the agents selected for this team-role and the subordinated team-roles: TR6, TR7. The Aggregation capability is a task-decomposer introducing two subtasks: Elaborate-items (TR6) and Aggregate-items (TR7). We note A^{TRi} as the agent selected to play Team-Role i. All the agents begin at the Start scene, and then:

1. A^{TR5} and A^{TR6} move from the Start scene to the Elab.Items (EI) scene; A^{TR5} adopts the Coordinator role (x:Coord), while A^{TR6} takes the Operator role (y:Op). Both roles are required to perform the scene, so the edge going from the first transition to this scene is a conjunction of them (x:Coord, y:Op).

2. A^{TR7} moves from the Start scene to the Ag.Items(AI) by taking the Operator role (z:Op) and waiting A^{TR5} at the AND transition placed between the

Elab.Items scene and the Ag.Items scene. A^{TR5} gets to that AND transition after playing the Coordinator role at the Elab.Items scene ($x : Coord$). The AND transition forces the incoming agents ($x : Coord, y : Op$) to synchronize before proceeding to the Ag.Items scene. As we can see in the picture, after crossing that transition, both agents continue playing their previous roles, A^{TR5} as Coordinator, and A^{TR7} as Operator.

3. Agents acting as operators can leave the performative structure just after finishing the scenes they have participated in. In particular, A^{TR6} and A^{TR7} can move to the End scene through an OR transition from the Elab.Items and the Ag.Items scenes respectively. However, the Coordinator role cannot abandon the performative structure until all the scenes have finished.

Teamwork is guided by a task-decomposer performative structure. Since some subordinated team-roles are also assigned to task-decomposers, new performative structures should be initiated when an agent assigned to a task-decomposer is requested by his coordinator. Therefore, the Teamwork process can be modelled as a nested structure of performative structures. There is one performative structure for each task-decomposer, starting from the top team-role.

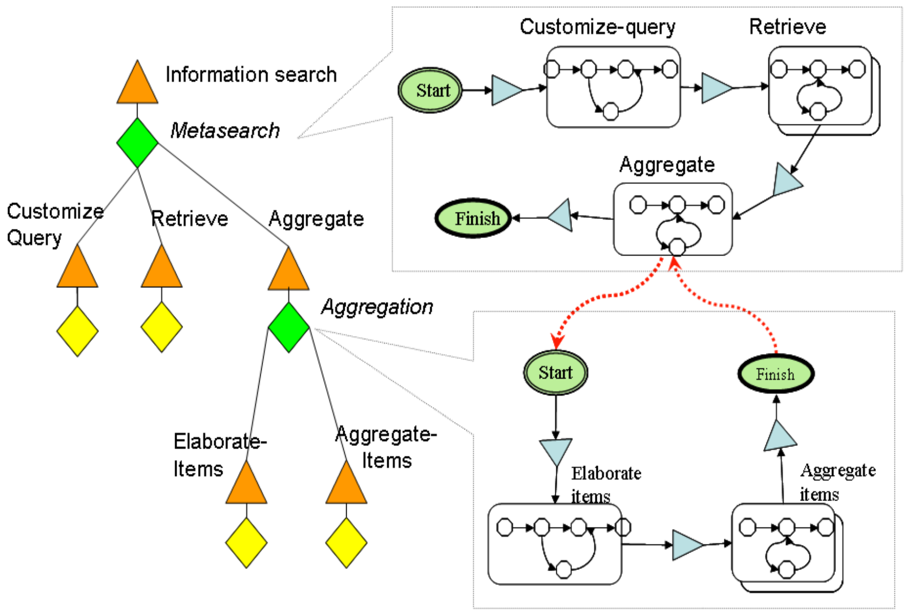


Figure 5.24: Teamwork model for a team

Figure 5.24 sums up the specification of the teamwork activity as a nested structure of performative structures. Notice that there is a performative structure for each task-decomposer in the task-configuration and one scene describes the interaction protocol required to delegate one task to another agent. The performative structures indicate which roles are required to play each scene, and the dependencies among scenes, e.g. some scenes must be finished before starting another scene, other scenes can be performed in parallel, or an scene can be instantiated multiple times in parallel. Moreover, the agent acting as the coordinator within a performative structure is as well holding the information about the team mates assigned to its subtask, specified within its team-role subteam (Definitions 5.1 and 5.2). The coordinator agent can initiate each scene at the right moment by contacting the agent or agents assigned to the corresponding team-role and following the selected scene. The agents playing some team-role must wait until a new scene is initiated by the coordinator. Moreover, a performative structure can include choice points that give the coordinator alternative paths to follow. It is a design decision to develop complex task-decomposers with many alternatives or to build many simpler decomposers with few alternatives or no alternatives at all.

The teamwork process follows the hierarchical structure of the task-configuration, decomposing a task into subtasks when there is a task-decomposer, and delegating some subtasks to other team members. The teamwork process starts with the team-leader (the agent assigned to the root task in the task-configuration) having to apply a task-decomposer. The team-leader starts the team-work process by following the performative structure that specifies the operational description of its task-decomposer. The team-leader engages in conversations with their subordinated agents in order to delegate them the subtasks specified in its task-decomposers. A task is delegated by following the scene specified by a team-role, providing the input for the data to the selected agent (as indicated by the selected agents feature of the team-role), and receiving the result for that task.

When a subordinated agent has to apply a task-decomposer itself, it does the same that the team-leader: delegates subtasks to selected agents and wait for the results, aggregate the results when opportune and send the global result to its own coordinator. The process of applying a task-decomposer follows the performative structure. Since some of the subtasks may be bound to task-decomposers too, a new performative structure must be carried over for each task-decomposer. The first performative structure (the one initiated by the team-leader) cannot finish until the new performative structures are finished. Therefore, performative structures are nested, a performative structure cannot finish until any performative structure under it finishes.

When a subtask is allocated to the same agent applying the task-decomposer, the scene associated to that task in the performative structure is skipped, since there is no need for communication. Instead, the agent solves the subtask himself by applying the required capability, realize that communicating with himself has no sense.

When a subtask is allocated to another agent, the agent applying a task-decomposer initiates the scene specified in the team-component associated to that subtask by sending the first message. For instance, if the scene specifies a

Request-Inform protocol, then the coordinator sends a request with the following information:

1. a team identifier;
2. a team-role identifier;
3. the input-data required by the subtasks associated to that team-role

All this information is included within an object of the sort Perform, as defined in the Teamwork ontology 5.6.

5.7 Extensions of the Operational Framework

The Cooperative Problem Solving process already presented has some limitations that arise when addressing runtime time dependencies among tasks and dynamic events altering the expected outcome of the Teamwork activity.

On the one hand, two common perturbations in the CPS process came from agent failures (i.e. an agent is unable to achieve a task) and communication errors (e.g. a message does not get to its destination). Since Team Formation can assign reserve agents, some times it is still feasible to resume the CPS process after an agent failure, without reconfiguration, by requesting reserve agents to play the associated team-role. However, sometimes there are no reserve agents to perform the unfinished tasks, and then a reconfiguration mechanism is required to look for agents equipped with alternative capabilities (i.e. other capabilities suitable for the task at hand and compatible with the problem requirements if possible).

On the other hand, some tasks may need or may benefit from information obtained at runtime in order to be configured. A task is configured by selecting a capability suitable for it, binding the capability to the task, and recursively configuring the subtasks of the capability when it is a task-decomposer. Sometimes, the selection of one capability or another may be improved or requires some information obtained at runtime. Therefore, the Knowledge Configuration process should be delayed for those tasks until the required information is obtained. In order to configure those tasks, we allow a capability to produce information to be used by the Knowledge Configuration process, such as a new precondition stated to be true, a new postcondition to be achieved, or a new domain model characterizing new domain knowledge (some capabilities may generate domain knowledge).

A more flexible CPS process is required to deal with such situations; therefore, we introduce some extensions to the CPS process that consist in different ways of interleaving the Knowledge Configuration, the Team Formation and the Teamwork processes.

5.7.1 Interleaving Teamwork, Knowledge Configuration and Team Formation

We consider three strategies that interleave Teamwork, Knowledge Configuration and Team Formation, 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.

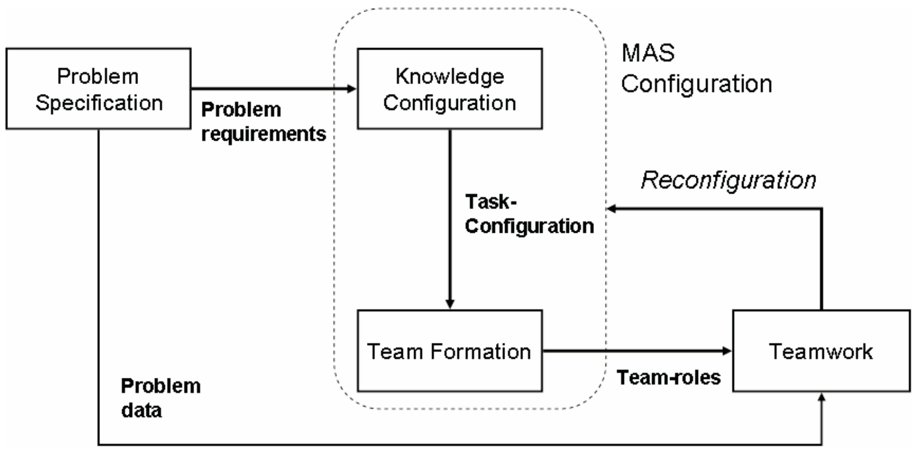


Figure 5.25: Extended model of the Cooperative Problem Solving process

Figure 5.25 shows an extended model of the Cooperative Problem Solving process capturing the notion of Reconfiguration. If the new capability bound to the task is a task-decomposer, then their subtasks must be further configured. A capability satisfying the global problem requirements is required, though a partial satisfaction criteria can be used instead to allow the Knowledge Configuration to succeed even when a fully satisfactory condition is unreachable. If the new capability bound to the task is a skill, then the reconfiguration ends there, otherwise the capability is a task-decomposer requiring a recursive configuration of the new subtasks.

The *Delayed-Configuration* strategy is used to hold the configuration of some task up until some event happens or some information is obtained.

Figure 5.26 shows a capability that performs a Propose-Critique-Modify method over the Information Search task, by decomposing it into three subtasks: P-Search (Propose-Search), C-Search (Critique-Search) and M-Search (Modify Search). The M-Search task is decomposed by the task-decomposer Modify-metasearch into four subtasks; the first of these tasks, Adapt-query, can be solved by two skills: Query-generalization and Query-specialization. The selection of one skill out of the two former skills depends on the result of the P-Search task: on the one hand, if P-Search obtains too many results then the capability skill

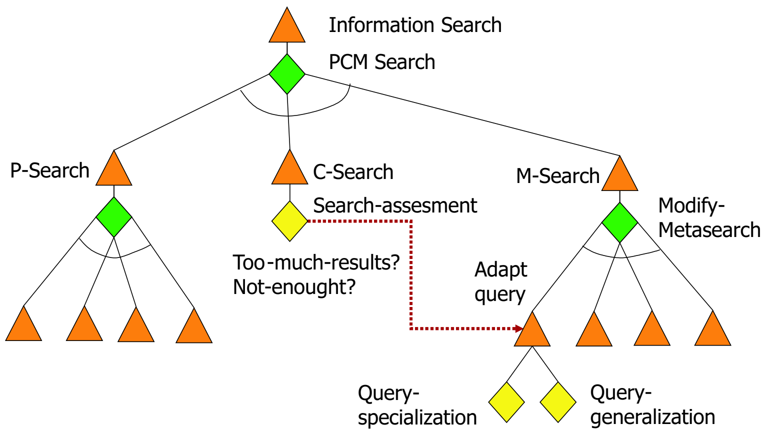


Figure 5.26: Propose-Critique-Modify Search

Query-specialization is preferred; on the other hand, if the are slender results that are considered not enough then Query-generalization is better. Otherwise, the number of results is considered adequate and the task M-Search is avoided.

In order to use the information obtained in runtime at the Knowledge Configuration process, we have specified the Query-generalization and Query-specialization capabilities as having different, incompatible postconditions: Query-generalization includes the postcondition Generalize-query, while Query-specialization includes the postcondition Specialize-query. There is only one capability suitable for the task C-Search, Search-assessment. Search-assessment brings about one of the two former formulae according to whether there are too many results for the query, or there are not enough results as to be useful. Furthermore, if the number of results obtained for the task P-Search is considered adequate, Search-assessment outputs a different formula expressing that condition, so as to allow the coordinator realize the task M-Search can be omitted.

The coordinator of a task specified as requiring a Delayed-Configuration, must be aware of the conditions in order to to interrupt and resume the Teamwork process when required, and perform the necessary actions to assure a new Knowledge Configuration process is initiated using the new conditions bring about in runtime. Similarly, the coordinator of a task that cannot be achieved using the current configuration must initiate a new Knowledge Configuration process in order to find an alternative task-configuration.

In both cases, after a new Knowledge Configuration process is over a new Team Formation process is required to allocate the new tasks to a new team. The resulting team acts as a subteam of the original team, taking responsibility of the team-role corresponding to the task that has just been configured.

In addition, other situations characterized by very dynamic environments may benefit of a systematic *delayed configuration* strategy, because it has no sense to completely configure a task in advance in such a situation, or tasks will probably require a reconfiguration too often. We call this strategy *Lazy Configuration*. The idea of the *Lazy Configuration* strategy is to avoid configuration whilst possible, configuring a task just when it is required to be solved. Lazy configuration is used to handle very dynamic environments that made profitable to gradually configure a task whereas the Teamwork process progresses. In our implementation of Lazy Configuration, the Knowledge Configuration process operates by configuring every time just one level of the task decomposition structure of a task-configuration. Using this strategy, when a task-decomposer is bound to a task, each subtask is bound a capability, but the newly introduced task-decomposers are not further expanded into new tasks. Once a task is configured one-level deep, the Teamwork process runs until a new task-decomposer has to be applied that introduces some new tasks to be configured, and then the Knowledge Configuration process should be performed again following the Lazy Configuration strategy, and so forth for each new task-decomposer going to be applied. A variation of the Lazy Configuration strategy is the introduction of variable deep levels during the Knowledge Configuration stage.

An interesting possibility is to perform the different activities of the CPS process simultaneously. After starting a Knowledge Configuration process using the Lazy Configuration strategy, perform Team Formation and Teamwork next, but instead of stopping the Knowledge Configuration process after starting Team Formation, continue with the Knowledge Configuration while possible, running in parallel with the Team Formation and the Teamwork processes, in such a way that when the Teamwork has to solve a task that is bound to a task-decomposer, the task will be already configured. We call this strategy *far-sighted* strategy.

The consequence of introducing these variations is a greater flexibility of the original model of the CPS process to handle different kind of scenarios, though the more flexible the configuration strategy is, the more communication activity is required.

5.7.2 Operational scenarios: dimensions and some prototypical scenarios

This subsection will draft a future work discussion on dimensions that may constrain the application of the ORCAS framework, and some typical scenarios that can fit well in the ORCAS framework.

The **autonomy dimension** deals with the degree of autonomy possessed by agents. Very autonomous agents are designed for distributed control approaches in which agents keep local control during most of the problem solving process. In addition to decide the commitment to a team-role, autonomous agents may prefer to decide by themselves the plans to use for achieving the goals of a team-role, i.e. configure a task by himself, and deciding when and whom to delegate a task. On the other hand, agents may prefer to delegate some activities of the cooperative process to specialized agents, like brokers and matchmaker agents. Team Formation and Teamwork strategies may adopt a wide range of strategies according to the degree agents keep local control during the problem solving process. While a distributed control approach is more appropriate for dynamic environments and real-time applications, a more centralized control is better suited for a service-oriented model of the CPS process. The ORCAS Operational Framework is neutral about the autonomy dimension. We are not specifying here which agents are responsible for configuring a task or forming a team. Later, in Chapter 6, an infrastructure for developing and deploying agents according to this framework is presented. This infrastructure is based on the electronic institutions approach, which adopts and external view and is based in the idea of standardized patterns of behavior called agent roles.

Distributed control approaches may be implemented over this infrastructure by equipping problem solving agents with the capabilities and the permissions required to play institutional roles: the Knowledge-Broker, responsible for configuring a task during the Knowledge Configuration process, and the Team-Broker, responsible for selecting and instructing agents during the Team-Formation process. Although we have initially defined the Knowledge Configuration process and the Team Formation process as being entirely completed before moving to the following stage, we have presented also some extensions of the Operational Framework that allow to interleave all the processes involved in the CPS process: Knowledge Configuration, Team Formation and Teamwork. Specifically, the combined use of the Lazy Configuration strategy during the Knowledge Configuration process and the adoption of the Knowledge-Broker and Team-Broker roles by problem solving agents covers most of the existing distributed approaches to team formation with autonomous agents.

The ORCAS implementation of an agent infrastructure (Chapter 6) aims to support agents developed by third parties to partake in the CPS process, and thus there are institutional agents equipped with the reasoning abilities required to configure tasks and form teams. However, it does not imply that control is centralized in a classical sense, it rather means that requesters and providers are mediated by institutional agents facilitating their work. From a service oriented or a component-based software development (CBSD) approach (e.g. "off-the-shelf" components), providers (problem solving agents) may be interested on cooperating with other agents and relying on institutional agents to carry out the Knowledge Configuration process and drive the Team Formation process. The point is that in these approaches the global interest represented by the problem requirements or the user preferences is usually favored against the interest of individual capability providers. The institutional agents included in the ORCAS infrastructure bring an added value to both the requesters and the providers of capabilities, freeing them of complex and computationally intensive tasks like configuring a task or selecting the members of a team. Therefore, this facility promotes a light-weight approach to agent development, and is oriented towards compositional software development approaches in which applications are composed rather than constructed, by reusing existing components (agent capabilities and domain knowledge).

The **openness dimension** deals with the capacity of integrating external agents and other components (like knowledge repositories, databases and Web services). Open agent societies allow external agents to joint the society on runtime, on a dynamic basis, without having to recompile the system code. Openness is also related to the maintenance, extensibility and adaptiveness of a system, since a system can be modified or extended by incorporating or eliminating agents. Open agent systems are designed to facilitate the integration of heterogenous agents provided by different developers. However, the greater the openness, the greater the complexity.

A main topic concerning this subject is the management of ontology mismatches, that is to say, allowing agents to use different ontologies and handling the mapping required to translate concepts from one ontology to another. Connectors between components can be introduced to implement ontology mappings; like the *bridges* proposed in the UPML software architecture [Fensel et al., 1999]. In the current implementation of ORCAS we are using a common ontology to avoid ontology mismatching and focus on other aspects such as the coordination of agents. Nonetheless, the ORCAS Abstract Architecture is well suited o introduce such kind of components, indeed, because of the conceptual decoupling of tasks, capabilities and domain models. In ORCAS connectors could be inserted between capabilities and domain-models, and between tasks and capabilities as well. The use of connectors in ORCAS would has two dimensions: a knowledge-level specification, which allows to match components specified with different ontologies; and the implemented counterpart, which allows semantically (or syntactically) heterogeneous agents to interoperate during the Teamwork process.

To sum up, both the configuration and the coordination of agent teams can be distributed using the same basic model of the Cooperative Problem Solving process. Current research on coalition formation algorithms use distributed algorithms to deal with the combinatorial nature of this class of problems. Optimal anytime coalition structure generation algorithms has been devised [Shehory and Kraus, 1998, Sandholm et al., 1998, Larson and Sandholm, 2000]. Some minor modifications of the ORCAS framework are required to support a distributed approach to the Knowledge Configuration and the Team Formation processes. In a distributed scenario, agents should consider both task-dependencies and problem requirements when configuring a task. Since independent agents have a partial view of the problem, cooperation and coordination with other agents to look for an optimal global solution would be required.

We consider now some prototypical scenarios that may fit into the ORCAS framework for the Cooperative Problem Solving process: the *Agent Factory* model, the *Service Orchestration* model, and the *Contractual Agent Society.*

The *Agent Factory* model is based on a notion of production factories, and has been proposed as a design pattern by the Object Oriented Programming (OOP) community. The idea is to assemble existing components to build customized solutions. This model fits well with the "Off-the-Shelf" Components approach to software development and shares some similarities with the Software Configuration community. Concerning agents, the Agent Factory model can be applied to build agents or teams on-demand, according to some existing requirements, rather than forming a team from a set of pre-existing agents. We consider two approaches to the Agent Factory model:

- building and assembling team-specific agents from elementary components;
- instantiating and coordinating generic agent types.

In the first approach, agents can be as complex as necessary in order to minimize the number of agents participating in a team, so as to reduce communication overhead. In the second approach, agents are pre-built, though they can be somehow parameterizable. While the first approach favors Teamwork, the second one speeds up Team Formation.

The *Service Orchestration* model refers to the activities required to select, compose and execute several Web services to achieve a global task. The Service Orchestration model proposed by the Semantic Web Services approach has similar goals and shares many similarities with the ORCAS framework when comparing services against capabilities. Semantic Web services can be conceptually described as capabilities in ORCAS. The DAML-S ontology defines the following aspects of a service: a *profile* that brings the information needed by service-seeking agents to determine whether the service meets its needs; an *process model* on how does the service works, which should facilitate service composition and monitoring; and the *grounding*, which specifies the way to invoke and interact with a service:

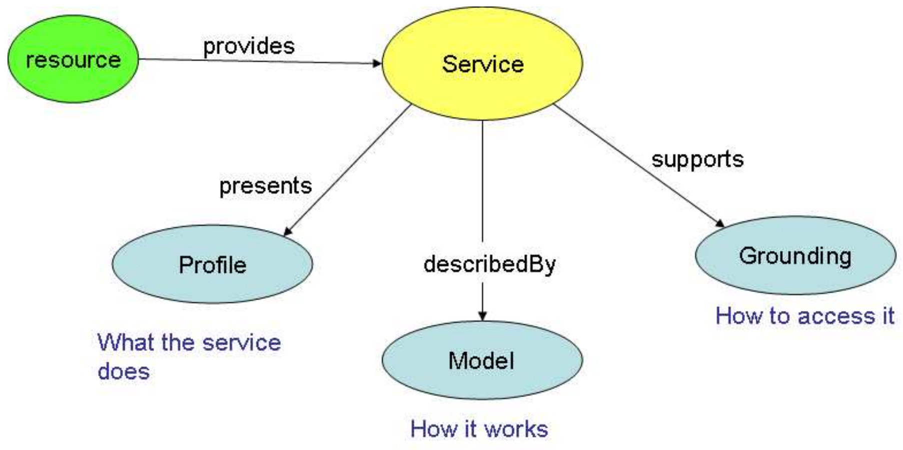


Figure 5.27: Service description according to DAML-S

Figure 5.27 shows the main elements of a service description according to the DAML-S ontology [The DAML-S Consortium, 2001]. All these aspects of a service have an equivalent in ORCAS:

DAML-S	Agent activities	ORCAS ACDL
Profile	Discovering (matchmaking)	Inputs, outputs and competence
Grounding	Invocation and Execution	Communication
Operational model	Composition and Interoperation	Subtasks and operational description

Table 5.1: Comparison of the ORCAS ACDL and DAML-S

- The aspects playing the role of the profile are provided by the knowledge-level description of capabilities.
- The purpose of *communication* of a capability is equivalent to the grounding of a service.
- The *operational-description* addresses the same features covered by the process model of a service.

Table 8.1 summarizes the relation between the features characterizing a capability in ORCAS, the features proposed to describe agent-enabled semantic Web services in the DAML-S ontology [The DAML-S Consortium, 2001], and the kind of activities these features are required for [Bansal and Vidal, 2003, Bryson et al., 2002, Park et al., 1998, Payne et al., 2001].

Due to these similarities between the Semantic Web Services (SWS) approach and the ORCAS framework, we think the ORCAS framework could be easily adapted to work upon SWSs as the providers of capabilities. Moreover, the use of agent wrappers over SWSs will allow the full integration of SWSs and agents within the ORCAS infrastructure.

The proposed *Contractual Agent Society* (CAS) model [Dellarocas, 2000] relies on a contractual agreement strategy for trusting agent interaction. Both providers and requesters of a service must agree upon the conditions associated to the provision of a service. Both the requesters and the providers have to comply with the conditions established by the contract, moreover, the contract specifies also the consequences of violating those conditions. The idea of using contracts fits well with the electronic institutions approach, in fact, we are considering as future work the introduction of what we call "terms of commitment" (Chapter 8), as a mechanism to agree upon by team members when accepting a team-role. This feature remains for the future work. e-Commerce applications like supply chains, auctions and e-markets are all based on contracts, therefore these applications are good candidates to apply the ORCAS framework.

5.8 Conclusions

Starting with a general set of requirements, we have stated the operational information to attach to the knowledge-level description of capabilities in order to become a full-fledged Agent Capability Description Language: the communication requirements of any capability, and the operational description of task decomposer capabilities. We have proposed electronic institutions concepts to specify such aspects of a capability. Specifically, communication requirements are specified as scenes —interaction protocols— and dialogic frameworks. Moreover, performative structures are used to specify the operational description of a task-decomposer.

The ORCAS infrastructure has been designed and implemented as an electronic institution. Actually, the ORCAS infrastructure can be seen as a meta institution where dynamic problem-solving institutions are configured on-the-fly, according to stated problem requirements. Some elements from the electronic institution formalism have been incorporated as components of the ORCAS Agent Capability Description Language. These elements —scenes, dialogic frameworks and performative structures— are defined for each capability, and are composed during the configuration of an agent team, which is done in two steps: first a task-configuration is obtained that specifies the competence required for a team to comply with stated problem requirements, and second, a team of agents is formed according to the task-configuration and ensuring that all the agents involved can interoperate by sharing a common institutional framework: communication language, ontologies, and interaction protocols (scenes).

Chapter 6

The Institutional Framework

This chapter describes an open agent infrastructure to develop and deploy MAS according to the ORCAS framework. This infrastructure is an electronic institution where problem solving agents meet to form teams and solve problems on-demand, according to the ORCAS model of the Cooperative Problem-Solving process.

6.1 Introduction

This chapter presents a particular implementation of the Knowledge Modelling Framework and the Operational Framework as an electronic institution, which is called the ORCAS e-Institution. The ORCAS e-Institution is an infrastructure for developing and deploying cooperative Multi-Agent Systems that supports both providers and requesters of capabilities along the different stages of the CPS process.

We have already presented a model to configure a MAS at two layers: the knowledge layer, called Knowledge Configuration, and the operational layer, called Team Formation. Moreover, we have presented a framework for the execution stage of the CPS process, that we call Teamwork.

This chapter describes an open agent infrastructure designed to use the ORCAS KMF as and Agent Capability Description Language, according to the two layers configuration model. The goal of this infrastructure is to allow the development and deployment of open, reusable and configurable Multi-Agent Systems:

- *Open:* agents can be created in multiple programming languages and interface with existing legacy systems.
- *Reusable:* tasks and capabilities are declared in a domain-independent way using its own domain-independent ontologies

- *Configurable on-demand:* components (capabilities and domain knowledge) are selected according to problem requirements and user preferences

During the rest of the chapter, and after a general overview of the ORCAS e-Institution in §6.2, we are going to review its dialogic framework §6.3, performative structure §6.4, and the communication scenes §6.5, in this order.

6.2 Overview of the ORCAS e-Institution

The ORCAS e-Institution acts as a mediation service for both clients and providers of capabilities. Both requesters and providers of capabilities are ruled by well defined interaction protocols (scenes), and mediated by institutional agents that reason about application tasks, agent capabilities and problem requirements using the ORCAS KMF as the Agent Capability Description Language. The ORCAS e-Institution is used to configure a MAS for a particular problem, which involves all the stages of the Cooperative Problem Solving process as described in the Operational Framework: Knowledge Configuration, Team Formation and Teamwork. The configuration of the MAS includes the Knowledge Configuration and the Team Formation process, and the result is a customized team of agents that is tailored to solve a specific problem according to its requirements. ORCAS teams are created and instructed to solve specific problems by obtaining a knowledge level configuration (a task-configuration) in terms of goals to achieve (represented as tasks), the competence (the capabilities) required by team members to achieve those goals, and the domain knowledge (satisfying the assumptions of the selected capabilities).

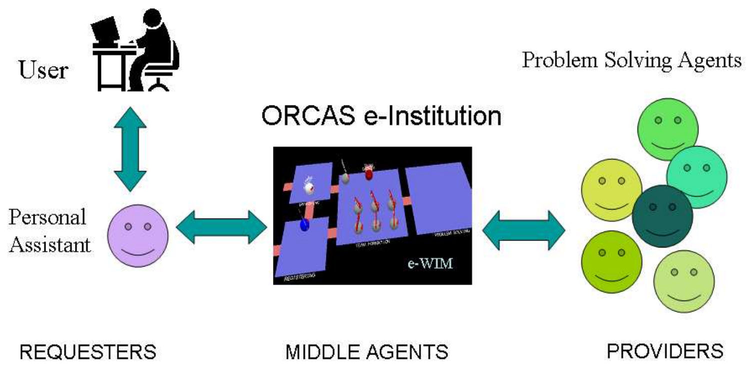


Figure 6.1: The ORCAS e-Institution as a mediation service between requesters and providers of capabilities

The ORCAS institutional framework is based on a client-server architecture extended with the notion of middle-agents [Decker et al., 1997b]. This model involves three kind of agents: providers, requesters and middle agents.

- *Providers:* agents providing specific capabilities to solve application tasks. In our architecture we call them Problem-Solving Agents (PSA).
- *Requester:* human or software agents requesting to solve a problem. We have included a class of agent called Personal Assistant to act on behalf of a human user. The Personal Assistant frees users of knowing the technical details needed to interact with other agents.
- *Middle agents:* agents mediating between requesters and providers. These agents are responsible for finding providers and instructing them to solve a problem. We consider three classes of middle agents: *librarians*, *knowledge-brokers* and *team-brokers*. *Librarians* act like a "yellow pages" service. They are dynamic repositories of agent capabilities, providing the link between the knowledge layer (task-configurations) and the operational layer (agent teams). *Knowledge-brokers* are able to obtain configurations of the MAS in a declarative manner, according to a problem specification. *Team-brokers* deal with the operationalization of a configuration, which consist in forming a team of problem solving agents with the capabilities required by a task-configuration.

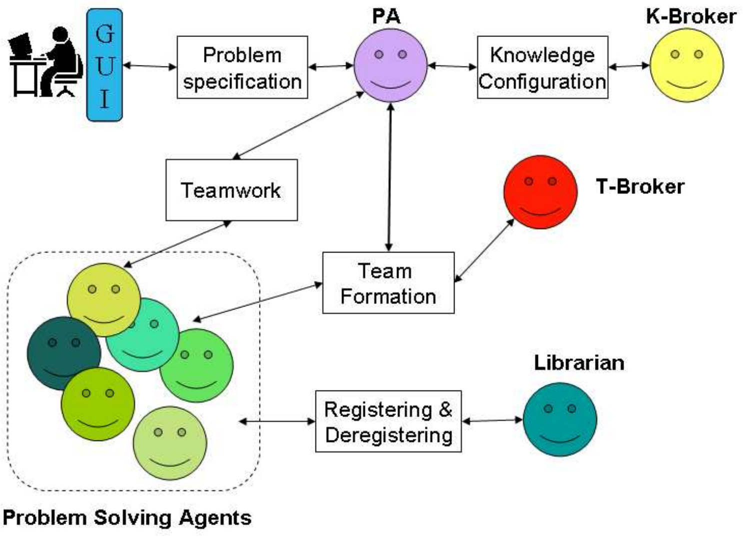


Figure 6.2: ORCAS e-Institution: main agent roles and activities where they are involved

Figure 6.2 shows the main agent roles that can be played by agents participating in the ORCAS e-Institution: Personal Assistant, Problem Solving Agent, Knowledge-Broker, Team-Broker, and Librarian. Bidirectional arrows represent the communication scenes where agent participate to carry out the different stages of the CPS process: Registering/Deregistering, Problem specification, Knowledge Configuration, Team Formation and Teamwork.

The rest of the chapter explains in detail the ORCAS e-Institution, using the ORCAS KMF as the ACDL and supporting all the stages of the ORCAS model of the CPS process (Knowledge Configuration, Team Formation and Teamwork). Since many elements of the electronic institutions formalism used within this chapter are described in Chapter 5 (The Operational Framework), we will refer the reader to the appropriate sections within that chapter when introducing each element of the ORCAS e-Institution.

6.3 Dialogic Framework

The dialogic framework specifies the ontological elements and communication language (ACL) employed during agent interactions. A dialogic framework can be defined either globally, for the entire e-institution, or using one dialogic framework per scene.

There are two kinds of agent roles in an electronic institution: *external* roles, than can be played by external agents, and *internal*, institutional roles. In our case, we consider as external roles the ones played by both requesters and providers of capabilities, namely the Personal Assistant (PA) and the Problem-Solving Agent (PSA) roles. However, middle agents are defined as internal roles, belonging to the institution: Librarian, Knowledge-Broker and Team-Broker (T-Broker).

There are five main agent roles in the ORCAS e-institution, namely Personal Assistant (PA), Librarian, Knowledge-Broker, Team-Broker, Problem-Solving Agent (PSA), plus two subroles of the PSA role: Coordinator and Operator.

1. *Personal assistant (PA):* An agent acting on behalf of a human user. This agent is responsible for mediating between the user request and the services offered by the application. The PA is able to specify problems in terms understood by the Knowledge-Broker, that is to say using the ORCAS KMF meta-ontology and Feature Terms as the object language. Furthermore, the PA is able to interact with the Team-Broker during the Team Formation process, and to start the Teamwork activity once the team is formed.

2. *Librarian:* This agent holds the knowledge descriptions of the reusable components: tasks, capabilities, domain-models and ontologies. The library can be dynamically updated or extended with new component descriptions by following a registering/deregistering procedure. Therefore new agents can enter the system by registering their capabilities to the Librarian using the ORCAS KMF as ACDL. The Librarian agent is a dynamic repository of ORCAS KMF components, allowing other agents or humans to query about them. Hence, the Librarian can be used as a "yellow pages" service, just keeping an up-to-date record of the association between ORCAS components and the agents that registered them. In the ORCAS e-institution, the Librarian is queried by the Knowledge-Broker to

know which are the components available to the Knowledge-Configuration process.

3. *Knowledge-Broker:* The purpose of the Knowledge-Broker is to configure an application for a user (represented by the PA) requesting to solve some problem. The Knowledge-Broker uses the specification of a problem as an input to generate a task-configuration: a structure of ORCAS components matching the problem specification, using tasks, capabilities and domain-models. The Knowledge-Broker is thus the responsible for performing the Knowledge Configuration process.

4. *Team-Broker:* The purpose of the Team-Broker is to operationalize a task-configuration by forming a team of problem solving agents. A team is a group of agents committed to solve a problem together, according to a task-configuration. A team is formed by finding and selecting agents with the required capabilities, and instructing them to cooperate in solving a problem together.

5. *Problem-Solving Agent* (PSA): This role is adopted by the agents willing to provide some capabilities. Problem-Solving Agents can join or leave the system dynamically, just registering or deregistering their capabilities to the Librarian. This is a simple way to make the Librarian aware of the capabilities available in the system at any moment.

Figure 6.3 shows the agent roles defined by the ORCAS e-Institution. Notice how they are organized according to the two kinds of agent relationships established by the electronic institutions formalism: SSD (dotted lines), and a partial order relation defined as a subclass relationship ($\succeq$).

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. This constraint reinforces the trust of external agents on the institution.

Moreover, we have defined two subclasses of the PSA role, which can thus be adopted by any agent playing a PSA role: the *Coordinator* and the *Operator*.

- The *Coordinator* role is adopted by an agent playing a task-decomposer team role and having to delegate some subtask to other agents. During the Teamwork process, any agent having to apply task-decomposer must initiate the scene specified each subtasks within its team-role sub-team. Meanwhile, the agents assigned to team-roles associated to a subtask adopt the operator role. The coordinator is responsible for distributing problem data and intermediate results among the operator agents, and is responsible for coordinating them. An agent acting as coordinator follows the performative structure that specifies the operational description of its task-decomposer, initiates communication scenes to interoperate with its subordinated agents, and performs any intermediate data processing when required.

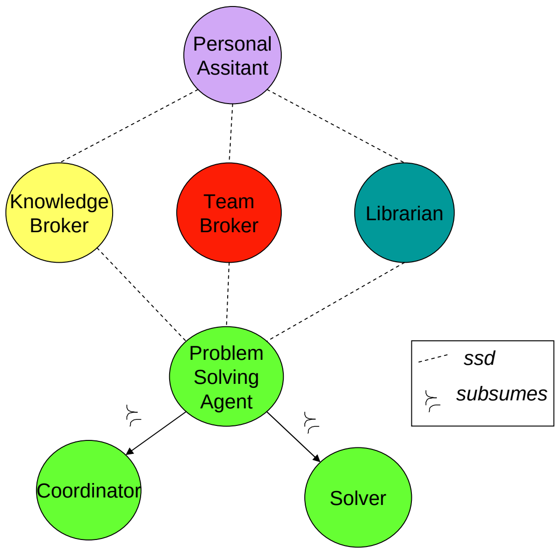


Figure 6.3: ORCAS e-institution roles

- The *Operator* role is adopted by the agents willing to perform a subtask of a task-decomposer. The agent assigned to the task-decomposer adopts the coordinator role and is the initiator of the scenes used to interact with every operator. The operators receive the data they require to solve their tasks from the coordinator, perform the capabilities assigned to their tasks, and send back the results to the coordinator.

Both the Coordinator and the Operator roles are dynamically assigned to PSAs, that is to say, they are not assigned to an agent when it enters the institution, but during the Teamwork process. An agent playing the PSA role switches to either the Coordinator or the Operator roles during the Teamwork stage, according to the following rules: when an agent has to communicate with another agent playing a subordinated role, the first agent adopt the Coordinator role and initiates a scene in which the second agent takes the Operator role; complementarily, if an agent has to communicate with an agent he is subordinated to, he adopt the Operator role on-demand, just after receiving a message from another agent adopting the Coordinator role.

A PSA can be playing both the Coordinator and the Operator roles simultaneously at different scene instances. Since a subtask may itself be bound to a task-decomposer, the agent selected for such a subtask has to act as the Operator with respect to the agent he is subordinated to, but before finishing that interaction, the same agent may partake in other scenes with its subordinated agents as operators, and acting himself as coordinator. Figure 5.18 shows an example of a team where this condition will happen: the agent selected for Team-role 5 (TR5) is assigned the task Aggregate, which is a subtask subordinated to TR1 (task Information-Search); thus the agent in charge of TR5 has to play the operator role with respect to the agent in charge of TR1. But while engaged in a scene with the agent playing TR1, the agent playing TR5 has to engage a scene acting itself as coordinator, while the agent playing TR7 acts as operator, since TR7 is assigned the task Aggregate-items, which is subordinated to TR5.

The Coordinator and the Operator role are not under a SSD relationship, because an agent can play, as exemplified above, both roles at the same time. As we explain later concerning Teamwork, during the Teamwork scene 6.5.4, team members can engage new problem solving scenes when needed, adopting either the Coordinator or the Operator role according to the subordination relations established by the hierarchical structure of a team-configuration **??**.

```
(define-dialogic-framework ORCAS_e-Institution_df as
    ontology = (KM-Ontology Teamwork-Ontology Brokering-Ontology)
    content-language = NOOS
    illocutionary-particles = (request inform accept refuse)
    external-roles = (PSA PA)
    internal-roles = (Librarian T-Broker K-Broker)
    social-structure ((PA ssd PSA)))
```

Figure 6.4: ORCAS e-institution dialogical framework

Figure 6.4 shows the dialogic framework of the ORCAS e-institution. The ontology contains the vocabulary and the concepts used by agents to communicate when other agents participating in the institution. Any ORCAS e-Institution comprehends at least three ontologies, the Knowledge-Modelling Ontology, the Brokering Ontology (describe later, in §6.5.2), and the Teamwork Ontology, which contain all the concepts required by the different roles of the ORCAS e-Institution to participate in the institution. Moreover, any application of the ORCAS e-Institution adds library specific concepts that should be shared by the agents participating in that application, like the concepts included in the ISA-Ontology (D), which is used by the agent participating in the WIM application (§7). The content language is NOOS [Arcos, 1997], a reflective object-centered representation language designed to support systems integrating knowledge-modelling and learning.

In order to implement the ORCAS framework as an electronic-institution, we have added some concepts to the Teamwork ontology which are required by institutional agents to communicate at the different scenes of the ORCAS e-Institution. These concepts should be understood by both the external and the internal, institutional agents in order to interoperate. These concepts will be introduced as needed when describing the different scenes of the ORCAS e-Institution within this chapter. As we have introduced in Chapter 5, we use elements of e-Institutions to specify the operational description and the communication requirements of agent capabilities; however, these elements should be distinguished from the ORCAS e-Institution.

On the one hand, the ORCAS infrastructure follows the original electronic institutions formalism ([Esteva et al., 2002b]) in that is is specified as a fixed structure of scenes (a performative structure), which are known beforehand.

On the one hand, the communication supported by an agent over a capability is specified as a scene (§5.4.2), while the the operational description of a task-decomposer is specified as a performative structure (§5.4.3), but these, these elements do not make up an electronic institution as conceived in the referred formalism; instead, they constitute a collection of incomplete performative structures (recall that scenes are not typed) that are completed (by selecting the scene types) and composed dynamically during the Teamwork process following a nested structure. These structures can be seen as special class of electronic institutions that we like to call dynamic e-Institutions, since they are configured on-the-fly, according to the requirements of the problem at hand.

Actually, since the nesting of performative structures occurs within the Teamwork scene in the ORCAS e-Institution, the ORCAS e-Institution can be seen as a meta-institution that defines the environment for the execution of dynamic, throw-away e-Institutions. This idea introduces becomes a new topic we put off as deserving further research (Chapter 8).

6.4 Performative structure

The performative structure of an ORCAS e-institution represents the network of interaction scenes, together with the relationships among scenes, that describe the paths followed by agents playing some role in the institution. The ORCAS performative structure contains four communication scenes, plus the Start and the End scenes, from where agents enter and exit the institution. The main scenes of this institution are the following:

1. *Registering scene:* where a PSA can register its capabilities to the Librarian in order to become available for the CPS process, as well as deregister when leaving the institution. The registering/deregistering process keeps the Librarian with an up-to-date description of the capabilities available at the system at any moment.

2. *Brokering scene:* this scene describes the pattern of interaction needed to obtain a task-configuration by the Knowledge Configuration process. The participants are a PA requesting to find a task-configuration, the Knowledge-Broker responsible for building the task-configuration, and the Librarian, holding the current description of the capabilities available in the system (the library of problem-solving components).

3. *Team Formation scene:* this scene describes the communication protocol for selecting and instructing the members of a team. The agents involved

are the PA, that holds the task-configuration obtained by the Knowledge-Broker at the Brokering scene, the available Problem-Solving Agents, that wait for team-role proposals to join a team, and the Team-Broker, responsible for selecting the team members and instructing them on the way to cooperate and coordinate with other team mates.

4. *Teamwork scene:* finally, once a team of agents has been formed and instructed to cooperate, the team-mates go to this scene to solve the problem in a cooperative way, using the information provided to them during the Team Formation scene by the T-Broker. The agents involved are the PA, holding the input data for the problem at hand, and the selected team members (PSAs).

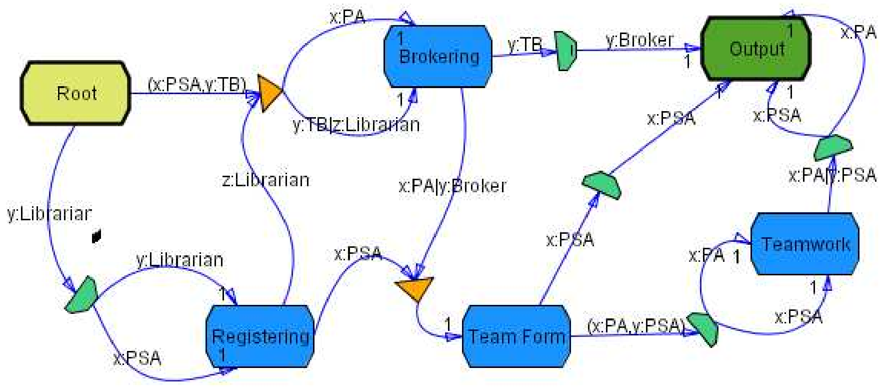


Figure 6.5: Performative structure of the ORCAS e-institution

Figure 6.5 shows the graphical representation of the ORCAS e-Institution performative structure. In addition to the four main scenes (Registering, Brokering, Team Formation and Teamwork) there is a root scene and an output scene as the initial and final scenes respectively. The role-flow policy is represented by the edge labels and transitions between scenes. Notice a PSA has to move first to the Registering scene, and only then can a PSA move to the Team Formation scene to wait for team-role proposals. The PA must start in the Brokering 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.

6.5 Communication scenes

This section describes the different communication scenes in the performative structure of the ORCAS e-institution: registering and deregistering (§6.5.1), brokering (§6.5.2), Team Formation (§6.5.3) and Teamwork (§6.5.4).

6.5.1 Registering scene

The Registering scene describes the communication activities used to allow the Librarian to be aware of the agents available in the system at any moment, and the capabilities they are equipped with.

Actually, there are two complementary activities, registering and deregistering. On the one hand, PSAs willing to join the ORCAS e-Institution must register their capabilities to the Librarian agent; on the other hand, PSAs willing to exit the institution must inform the Librarian they are leaving, so as to allow the Librarian update the library.

The Registering scene follows a request-inform protocol. When a PSA enters the agent platform, it sends a "register" message with the set of capabilities it is equipped with. The Librarian builds a table with the bindings between capabilities and agents, and keeps it updated by tracking the registering and deregistering activities of PSAs.

Figure 6.6 shows the registering scene specified as a request-inform protocol. The scene starts with a PSA requesting the Librarian to register a set of capabilities (transition 1). The Librarian can then accept (transition 2) or refuse that request. If accepted, the Librarian informs the PSA whether the requested capabilities have been successfully registered (transition 3) or there was some problem (transition 5).

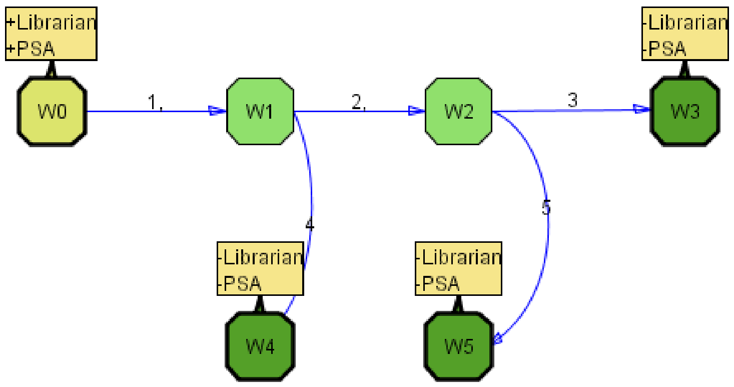
6.5.2 Brokering scene

The purpose of the Brokering scene is to allow the PA to obtain a task-configuration satisfying specific problem requirements. Recall that a task-configuration is obtained through a Knowledge Configuration process (§4.4), which takes a specification of problem requirements and a specification of the components in the library (tasks, capabilities and domain-models) as inputs, and produces a task-configuration as output.

There are three roles participating in the Brokering scene: the Knowledge-Broker, the Personal-Assistant and the Librarian.

The Knowledge-Broker (K-Broker) is the role played by the agent responsible for the Knowledge Configuration process, which is implemented as a search over the space of possible configurations, where the problem requirements are constraints to be satisfied by the configuration.

The Personal Assistant (PA) role represents the user and deals with all the human-computer interaction. The PA is defined as an external role, since only the communication layer of the PA are domain-independent. In general, the PA has a common social layer that defines the acceptable behavior of the PA



1.	(W0 W1)	request (?x PSA) (?y Librarian) register(?capabilities))
2.	(W1 W2)	accept (!y Librarian) (!x PSA) register(!capabilities))
3.	(W2 W3)	inform (!y Librarian) (x! PSA) register(!capabilities))
4.	(W1 W4)	refuse (! Librarian) (!x PSA) refusal(?reason))
5.	(W2 W5)	inform (!y Librarian) (!x PSA) failure(?error))

Figure 6.6: Specification of the Registering scene

within the institution, while there is an application specific and even a user specific layer dealing with the particularities of a specific application or user. In the ORCAS implemented e-Institution, the domain-independent layer of the PA is separated from the application specific details, which are implemented separately, as as pseudo-agent that manage the graphical interfaces. There are several interfaces, some of them are general, domain-independent, and oriented towards expert users (e.g. the knowledge engineer), whilst others are application specific, like those included in the WIM application to interact with the end-user (Chapter 7). An example of a domain-independent interface is shown in Figure 4.19, while examples of domain-dependent interfaces are depicted in Figure 7.17 and Figure F.4.

Moreover, the PA role defines just the communication requirements for an agent holding the specification of a problem to be solved so as to interact with the rest of agents in the institution. The basic function of the PA is to mediate between the user and the institution so as to relieve the user of holding any knowledge about how to locate and interoperate with other agents. Rather than including domain-specific knowledge as part of the PA, we preferred to implement a generic, domain-independent PA, and include domain-dependent knowledge as part of the interface (interfaces are implemented as pseudo-agent that can communicate with an agent using the agent communication language). To sum up, the PA brings an added value to the application tasks by providing customization services, and domain-informed support to the decision taking during the Problem Specification process.

The problem specification process is skipped here, since it is performed outside the institution; however, we assume that the PA is holding a complete specification of a problem to be solved, represented using the ORCAS ACDL and the vocabulary from the application ontology (e.g. the ISA-Ontology in WIM).

The Brokering scene begins when the PA sends a request message to the Knowledge-Broker (K-Broker) containing a specification of problem requirements. The K-Broker is an agent that is able to obtain a task-configuration satisfying specified problem requirements on-demand, out of the component specifications hold by the Librarian.

Once the K-Broker receives a request from the PA, it asks the Librarian to get an updated version of the components registered in the library at the moment, searches a task-configuration satisfying the problem requirements, following one of the three configuration strategies described in §4.4.4.

The K-Broker can ask the Librarian for the entire library at the beginning, or it can ask the Librarian several times, requesting only those components satisfying a matching criteria so as to retrieve only useful specifications. For instance, when going to bind a capability to a task, the K-Broker can ask the Librarian for just those capabilities matching that task.

The concepts required to participate by the PA and the K-Broker to participate in the Brokering scene are included in the Brokering ontology Figure 6.7.

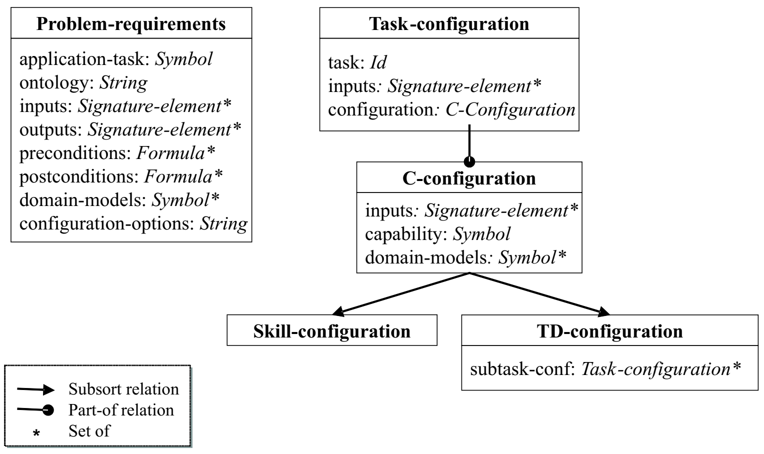


Figure 6.7: Broker Ontology

The input for the Knowledge-Configuration process is a specification of *problem requirements* composed of a) the name of the task to be achieved, b) the pre-conditions that are established to hold, c) the postconditions that have to be hold when the task is achieved, and d) the domain-models that are available for achieving the task. The outcome of the Knowledge-Configuration process is a task-configuration: a tree of triplets containing a task, a capability suitable for that task, and one or more domain models satisfying the knowledge requirements of that capability.

Figure 6.8 shows the interaction protocol for the Brokering scene. The scene starts (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 specification 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 (transitions 2, 4), or it can ask for a partial set of tasks or capabilities satisfying some matching criteria (transitions 3, 5). The purpose of these actions are to make the K-Broker work with an updated version of the library during the Knowledge Configuration process, while the use of one or another mode depends on the strategy adopted by the K-Broker. In the first case the K-Broker gets the entire library in one single interaction, while in the second case the K-Broker takes several steps to obtain all the required information, but can retrieve only the information that is strictly necessary to configure a particular task, rather than using the complete library. 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 specification to the PA (w_6), ranking those tasks according to the similarity measure defined in the context of the Case-Based Knowledge-Configuration strategy (§4.5). The PA chooses one task from that set and informs the K-Broker (transition 7) on the selected task and the configuration strategy to use. 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 obtaining a task-configuration satisfying the requirements, it sends a inform message containing the resulting task-configuration to the PA (tr. 8), and the scene ends.

Notice that the process may fail at several points, causing the scene to end without obtaining a task-configuration. The Brokering scene (Figure 6.8) includes a second final state that is reached either when the K-Broker cannot find a task satisfying the requirements of the problem (tr. 10), or when the K-Broker cannot obtain a task-configuration (tr. 9).

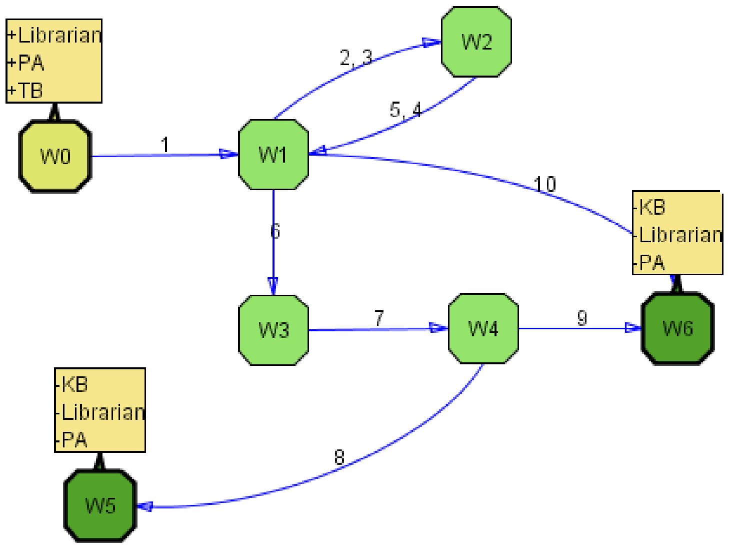
The Knowledge-Configuration process has been described in Section §4.4. This process is performed by the K-Broker during the Brokering scene, at state w_4. The Knowledge Configuration process is performed by the K-Broker as a state-space search in the space of partial configurations (Section §4.4.5).

6.5.3 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 build according to a task-configuration, and a group of agents is selected and instructed to play every team-role.

We have already described the Team Formation process as having three stages (§5.5): 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 for the



1.	(W0 W1)	request (?x PA) (?y TB) configure(Problem-Specification))
2.	(W1 W2)	request (!y TB) (?z Lib) retrieve-component(?specification))
3.	(W1 W2)	request (!y KB) (?z Lib) retrieve-library)
4.	(W2 W1)	inform (!z Lib) (!y KB) found(?components))
5.	(W2 W1)	inform (!y Lib) (!z KB) nothing-found)
6.	(W1 W3)	request (!KB null) (!PA null) choose-task(?task-list)) ( )))
7.	(W3 W4)	inform (!x A) (!y KB) configure(?task ?mode)) ( )))
8.	(W4 W5)	inform (!y KB) (!x PA) configured(?task-configuration))
9.	(W4 W6)	failure (!y KB) (!z PA) partial-configuration(?task-configuration)) ( )))
10.	(W1 W6)	failure (!y KB) (!z PA) no-task-found)

Figure 6.8: Specification of the Brokering scene

case of failures during Teamwork. Finally, the agents involved in the process are informed on the result of the team selection process, and the agents selected are instructed on the way they should coordinate with other agents during the Teamwork process.

The Team-Broker (T-Broker) is the responsible of 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 is able to reason about the component specifications and task-configurations, but it is specialized in forming teams, and has specific knowledge about operational descriptions that are useful to improve the team selection process (and so the performance of Teamwork). Section **??** describes the constructs of an operational description that can be used for that purpose: sequences, choices, parallelism, choices and multiple-instances (Figure 5.20).

First, the Team-Broker obtains candidate agents willing to play some team-role by sending team-role proposals and accounting for the agents accepting. Agents accepting a team-role proposal are considered as committing (by dialogue) to play that team-role, and are consequently expected to try to achieve the associated tasks when required during the Teamwork process. The Team-Broker will analyze the task-configuration to know which tasks should be solved by the team and which capabilities are required to solve each task. Using that information the Team-Broker can generate the team-model, one team-role for each task in the task-configuration, and propose those team-roles to agents willing to join the team.

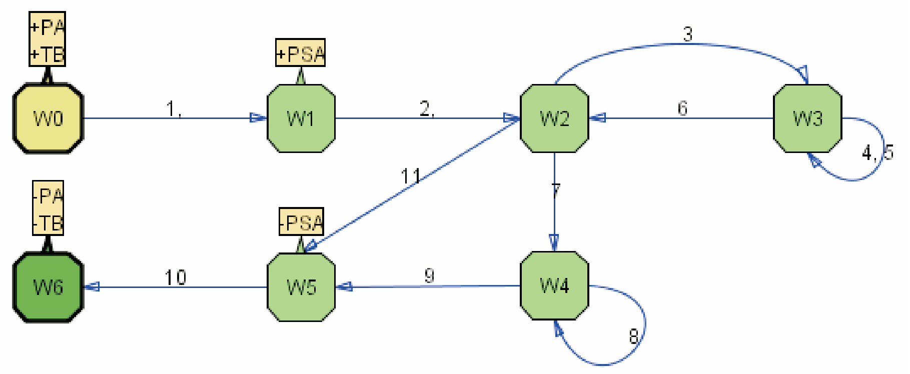
Problem-Solving Agents (PSA) can decide autonomously whether to accept or to refuse a team-role proposal. The Team-Broker waits until all the available agents have answered or a time out is reached, then, the Team-Broker decides among alternative agents for the same task which ones to select as members of the team.

Different algorithms and strategies can be used within the Team Formation scene to allocate tasks to candidate agents. It is possible for instance to select agents after each team-role proposal. Another strategy is first to obtain candidate agents for all the team-roles, and then to select agents for all the team-roles. Moreover, the infrastructure presented here is suitable for a wide range of selection strategies that can be embodied within the Team-Broker agent. The team selection strategy belongs to the Team-Broker internal decision-making strategies, and not to the institutional framework.

Figure 6.9 shows the specification of the Team Formation scene. There are three roles involved: the PA, the Team-Broker and the Problem Solving Agents.

The Team Formation scene starts at w_0 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 going to start (tr. 2).

The task-allocation and team selection processes follow an auction-like approach similar to the *contract-net* protocol [Smith, 1940]: team-roles are proposed to PSA agents willing to join the team (transitions 3); agents have then a



1.	(W0 W1)	request (?x PA) (?y TB) team-formation(?task-configuration))
2.	(W1 W2)	inform (!y TB) (all PSA) start-team-formation(?team-id))
3.	(W2 W3)	request (!y TB) (all PSA) commit(?team-role))
4.	(W3 W3)	accept (?z PSA) (!y TB) join-team(!team-role))
5.	(W3 W3)	refuse (?z PSA) (!y TB) join-team(!team-role))
6.	(W3 W2)	inform (!y TB) (all PSA) time-out(!team-role))
7.	(W2 W4)	inform (!y TB) (all PSA) start-team-configuration(?team-id))
8.	(W4 W4)	inform (!y TB) (?z PSA) commit(?team-role))
9.	(W4 W5)	inform (!y TB) (all PSA) finish-team-configuration(?team-id))
10.	(W5 W6)	inform (!y TB) (all PSA) finish-team-formation(?team-id))
11.	(W2 W5)	inform (!y TB) (all PSA) team-failure(?team-id))

Figure 6.9: Team Formation scene

limited period of time to accept (tr. 4) of refuse the proposal (tr. 6) before the time-out is reached (tr. 6).

If there are no candidate agents for all the team-roles, the team can not be formed at all. After performing several attempts without succeeding, the T-Broker informs participating agents of a team-failure (tr. 11) in order to call off the Team Formation scene and discard the ongoing team.

If there are candidate agents for all the team-roles, the task-allocation process succeeds and the T-Broker announces the beginning of the team selection process (tr. 7). During the team selection process, the T-Broker has to select the agents to play each team-role, and the agents to keep in reserve, using the information provided by both the operational description of task-decomposers (containing information about paralellism) and any application specific criteria considered. After being selected to play some team-role (w_4), team-members are informed on the team-roles they are assigned to and have to commit to (tr. 8). After finishing the team selection process, the T-Broker informs participating agents the team-selection is finished successfully (tr. 9).

As described in §5.3.1, team-roles are used to inform team-members about all they need to carry out a task within the team: the task to be solved, the capability to apply, the knowledge to use, and optionally, if the capability is a task decomposer, the information required to delegate subtasks to another team-members.

After selecting the agents to play every team-role, the identifier of the team and the information required to communicate with the team-leader (the top level team-role) are sent to the PA (tr. 10), and the scene ends. The PA does not require a complete team description because the information on each specific team-role has been submitted to each team member during the Team Formation scene.

6.5.4 Teamwork scene

This section describes the process of solving a problem by a team of problem solving agents (PSAs), once the team has been formed and instructed during the Team Formation stage, in such a way that the specified problem requirements are met.

The already introduced scenes (Registering, Brokering and Team Formation) are single scenes. However, the Teamwork scene is not specified by a single scene, though there is an initial scene that is used by the PA to request the team-leader to begin teamwork, provide the team-leader with the data for the problem and wait for the results. Nonetheless, the teamwork activity is not reduced to this initial scene, but will follow a nested structure of performative structures, one for each task-decomposer team-role (§5.6). Moreover, the scenes to be used within these performative structures are not predefined, but have been selected during the Team Formation scene from the set of communication scenes supported by the two agents involved in any task, one playing the Coordinator role, and another one playing the Operator role (§5.4.2).

The Teamwork process is initiated by the PA once the Team Formation has succeeded. The Teamwork scene defines just the initial scene of the Teamwork process, involving two agent roles, the PA and the PSA roles. There is a single agent playing the PA role, while all the team-members play the PSA role. This scene is used by the PA to send the problem data to the team-leader and get the final result back.

The Teamwork follows a request-inform protocol; the PA sends a "request" message to a PSA playing the team-leader team-role, (the responsible for the root task within the corresponding task-configuration). This message contains a team-identifier, a team-role identifier corresponding to the team-leader, and the input data for the problem at hand.

The team-role identifier is required by the team-leader to check up whether it is committed to that team-role, and to retrieve the information associated to that team-role so as to carry out the task associated to it. Since a PSA can participate in different teams at the same time, a unique team identifier is also required to avoid ambiguity.

When a PSA receives a request from the PA, the PSA 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 capability. According to the type of capability assigned to a team-role a PSA can face two situations:

If the capability assigned to the team-leader's team-role is a skill, the PSA has just to apply 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. Therefore, the Teamwork activity involves only one scene and two agents, one agent playing the PA role, and another agent playing PSA 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 the team-role's subteam, which specifies the agents selected for each subtask, as well as the scene to be used to communicate 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, and several scenes to be played, one for each subtask to be delegated to another agent. Recall that there is one performative structure (an operational description) for each task-decomposer capability assigned to a team-role. The idea is that the teamwork activity can be modelled by a electronic institution whose primitive elements are assembled on-runtime, during the team-formation process. Therefore, the teamwork activity follows the performative structure of that institution, expanding to a new performative structure each time a new task-decomposer is applied. Agents adopt the coordinator and operator roles as required, according to whether they are attending a request from another agent (operator) or they are applying a task-decomposer and have to delegate some subtask to other agents (coordinator).

As team members applying skills finish their tasks, the results are sent back to the coordinators, which are responsible for obtaining the result for his task using the result of the many subtasks, and propagating his own result back to his respective coordinators, and so on, until the team-leader obtains the final result and sends it to the PA.