# A flexible facilitator-based cooperation framework

Paul-André Tourtier
INRIA, ACACIA project
2004 Route des Lucioles
F-06902 Sophia Antipolis, France
tourtier@sophia.inria.fr

December 15th, 1994

**Abstract**

The aim of this work is the design of a flexible framework allowing distributed, autonomous, heterogeneous agents to cooperate through communication and interaction. We propose a practical cooperation framework that provides a wide range of communication and cooperation methods including: peer-to-peer and client-server interactions, data sharing, use of facilitators, etc. Based on this framework, we present an implementation of a cooperation platform and illustrate its use for distributed problem-solving.

## 1 Introduction

The aim of this work is the design of a flexible framework allowing distributed, autonomous, heterogeneous agents to cooperate through communication and interaction. Actually, the long-term motivation for this work is part of a more general strategy that can be stated as follows:

1. try to understand the way humans interact and cooperate to solve problems;
2. propose a general framework for the modeling of collaborative systems;
3. propose an architecture for the design of collaborative interaction systems;
4. provide a flexible platform for the realization of multi-agent systems in distributed environments.

In the present paper, we will focus especially on the realization of such multi-agent systems.

Let us consider then an environment made of multiple software entities (agents) interacting with each other. In the following, we will make a few assumptions on the nature of this multi-agent community:

**heterogeneous.** There is no restriction on the type of agents involved (knowledge-based systems, databases, or other pieces of software such as conventional numerical programs), on the structure, or on the implementation of such agents (architecture, programming language).

**distributed.** Agents may be distributed on different, possibly distant sites. It is usually possible to identify a site to a machine and an agent to a process. We will assume here that the sites are connected through a communication network that can support reliable communication protocols.

**cooperative.** The agents perform a variety of tasks that lead them to interact with each other for their mutual benefit and, in some cases, to cooperate explicitly. Although agents may have their own interests, we will assume that they exhibit generally a benevolent, cooperative attitude.

**dynamic.** Agents may evolve through time and change their goals, their methods, and even their location. New agents may be inserted within the community; other may disappear or be replaced by newer versions. As a consequence, as the environment grows larger, agents may not have a complete or accurate knowledge of each others.

**structured.** On the other hand, we will assume that the world is non-chaotic and that the rate of change in the environment is slow enough to allow agents to have a reasonably stable view of it. For instance, if some agent plays a role in the community by providing some kind of service, other agents can reasonably expect, either that it will fulfill its role for (at least) a while, or that they will be informed of that change. For short, the main structure of the community should be stable enough for agents to model it and unpredictable changes, although possible, should be rather exceptional.

It has been shown that some of the issues involved in the design of collaborative systems include namely [Marick and al., 1994]:

1. defining how and what to communicate;
2. defining, organizing, and allocating tasks to the agents;
3. controlling the global behavior of the system;
4. allowing agents to use existing tools and bring new ones on-line as they become available;

Our goal is to define a generic platform that would ease the development of cooperative multi-agent systems.

## 2 Approach

Actually, cooperation can be accomplished through a variety of ways and most previous approaches have put emphasis on certain communication methods (asynchronous message passing, data sharing, use of special-purpose languages) or on a specific coordination strategy (contract net protocols, joint commitments, exchange of plans, etc.). Common approaches include the following[Bond and Gasser, 1988]:

- provide protocols for registering and calling a set of public services (client-server paradigm);
- use peer-to-peer, asynchronous communication mechanisms, and favor reactive-based behavior (actor paradigm);
- organize coordination through concurrent access to a shared body of knowledge (blackboard approach);
- provide a framework allowing transparent operations on remote objects (object broker);
- specify knowledge-level languages and protocols with well-defined semantics (knowledge sharing approach);

- define market-based protocols based on requests and offers (contract nets);
- use mediators to facilitate cooperation (centralized coordination).

Instead of trying to enforce a particular vision of cooperation, we envision a practical cooperation framework that would provide a wide range of communication and cooperation methods. Our approach can be summarized as follows:

1. Provide a flexible communication platform, allowing namely peer-to-peer and client-server interactions;
2. Do not impose a fixed communication language but provide tools for the design of new communication protocols;
3. Provide facilities for data sharing and cooperation by data refinement;
4. Design facilitators, i.e. specialized agents whose goal is to facilitate the communication and the cooperation of other agents;
5. Favor a dynamic organization of the collective activity, made by the agents themselves;
6. Design man-machine interfaces allowing human users to inspect, control, and participate directly to the activity of the software agents.

# 3 A facilitator-based framework

The notion of facilitator is a key point of our approach. In the next section, we present a few examples of such facilitators and show how they can help the other agents to cooperate. Please note however that the following list is not intended to be complete and that new kind of facilitators could be easily added to the framework.

## 3.1 The communication facilitator

The role of the communication facilitator is to help agents to locate each other and to communicate for their mutual benefits. This includes facilities such as:

**yellow pages.** The purpose of this service is to allow agents to locate other ones, based on a set of indices such as name, location, function, interest, or competence. This is a general-purpose search mechanism that can be used, for instance, to facilitate contract net protocols or other form of subtasking. The usefulness of this facility depends, however, of the good willingness of agents to register to their local facilitator, to provide accurate informations, and to update regularly these informations (see below for ways of enforcing such a policy).

**address-based routing.** If there is no direct communication links between two agents, agents can use this facility and ask the communication facilitator to route a message. For instance, the facilitator could select a path according to its knowledge of the network and of the current traffic, and ask the intermediate nodes to forward the message towards its destination.

**content-based routing.** Messages sent by an agent can also be routed based on interests asserted by other agents rather according to specific, prearranged address. When relevant information is published by some agent, it is picked up by the facilitator and forwarded on to the requesting agent as though told directly by the publisher. This allows agents, who don't know of each other's existence (which is likely to happen in large and dynamic environment) but who share common interests to establish communication [].

### 3.2 The blackboard facilitator

The role of a blackboard facilitator is to facilitate knowledge sharing among a set of agents by providing a concurrent access to a structured body of shared data, through a consistent interface. Its main functions are namely:

**data management.** The blackboard facilitator is responsible for the storage and the management of the shared data and for executing the operations that are requested by the agents. Classically, the shared memory is divided in several logical levels that can be redefined dynamically by agents. Each level contains a set of objects defined by a name (an identifier), a class, a set of attributes and, eventually, a set of links to other objects. Objects can point themselves to data of various nature, such as raw text, logical formulae, or hypermedia documents. The implementation details are not known by the agents: e.g. objects can be stored in a classical database, be represented in some high-level scheme based on first order logic, or be physically distributed across the network.

**communication management.** Each agent has to declare itself in order to be provided an access to the shared data. Agents can then send requests that will be treated concurrently by the blackboard according to its own management priorities and to the preferences expressed by the agents themselves. For that purpose, the facilitator manages various information on the personal status of connected agents (name, interests, rights, preferences, statistics, etc.)

**control.** Objects are also associated with management information such as: creator, owner, creation date, last modification date, allowed operations, access rights, etc. The blackboard facilitator provides a controlled access to these objects and may decide to hide or protect some information from unauthorized agents. An object can be defined by its owner either as public, either as shared by a subgroup of agents, or as private to its owner. Moreover, only the operations that have been declared for its class (e.g. read, modify, append, link, delete, execute) can be applied to an object, and different access rights may apply for different operations.

**action triggering.** Agents may request that the blackboard performs some action whenever some condition (a logical predicate) holds, e.g. "if the state of any object of class C changes, then inform agent X of the new state of that object". This is the equivalent to the notion of knowledge source rules in traditional blackboard systems. Using this feature, agents can be automatically informed of the events that are important to them (without having the burden to check regularly the state of the shared memory) and can react immediately to the new situation, if necessary.

### 3.3 The service facilitator

The role of the service facilitator is to favor agent coordination based on any kind of contracting.

**bidding** An agent that wants to provide or use a given service can declare itself as a server or a client of that service to the service facilitator. Here a functionality, or service, is simply defined

by a protocol, some access rights, and other properties such as: purpose, usage, time needed, constraints on input/output values, reliability, etc. The role of the facilitator is mainly to register the offers or demands of the various agents, to facilitate protocol-based transactions (e.g. contract net), and to answer service-related questions such as: "which agent is offering service S on machine M and to which conditions".

**control** Once a contract is engaged between two or several agents, the role of the service facilitator is to control that the clauses of the contract (the "rights and duties" of the agents) are fairly respected: e.g. that agents follow the rules of mutually aggreed protocols and that their commitments are actually realized by actions. Note that some kind of contract (role, task, etc.) may exist between an agent and the group itself; therefore, the service facilitator can also be used by the community as a way to enforce general policies.

# 4 RACE, a multi-agent platform

Based on the previous approach, we have designed and implemented a prototype of a multi-agent platform that we have christened RACE. Technically, our framework offers:

- a low-level *communication platform* that allows software agents running on distant sites to exchange informations through the network;
- an *agent-oriented language* (based on a C++ interpreter) used to program new software agents, either from scratch or by wrapping around existing applications;
- a predefined set of *facilitator agents* specialized in some aspects of inter-agent coordination (communication, data sharing, security);
- a set of *libraries* (with a C/C++ software interface) providing various communication and coordination facilities, including functions for communicating with the local and remote facilitators;
- a set of *tools* for compiling communication and data sharing protocols and for debugging purposes.

In the following, we will take a practical viewpoint and describe the framework from a programming perspective, i.e. by considering the tasks that have to be completed by an application engineer to design a multi-agent system.

## 4.1 Agent Identification

The first task is to identify the various agents involved and the nature of the communication protocols that they use.

Each agent is given an *address* defined by a name and by a location (a machine). For instance, an agent *foo* on a machine *bar* can be referred to by remote agents as *foo@bar* and by local agents as *foo@localhost*, or simply as *foo*. By convention, an agent can have different names but two agents that are located on the same machine cannot share the same name.

Each agent can react to, and send certain kinds of messages. Here, a type of message is defined by a name, a set of input parameters and, optionally, a return parameter (for synchronous communications). Messages that participates to the same functionality can be grouped into a *protocol*. Sender and receiver of these types of messages are respectively called clients and servers of that protocol. Therefore, an agent can be defined by a set of input or output *communication ports*, where each port is characterized by a given protocol that the agent uses either as a client, either as a server, or as both.

## 4.2 Communication Platform

The next job of the programmer is to define the very nature of the data that are exchanged between the agents. In the current framework, various protocols can be easily defined thanks to:

- a C-based *specification language* (a subset of the C language) used to describe the types of data that are exchanged between the agents. A protocol is defined by a set of functions, each function representing a type of message (name and input-output parameter types). Basic data types are provided (void, string, float, integer) but it is also possible to define complex new types (array, structure, pointer, etc.);
- a *generation tool* that compiles such protocols into three pieces of code that have to be linked with an agent' s own code: a client interface (used to receive messages), a server interface (used to send messages) and a communication stub (that supports encoding, transmission, and decoding of the message across the network). This implementation is based on the XDR protocol for maximum portability.
- a *shared library* of C/C++ functions that applications must be linked with. Written in C/C++, our implementation is based on efficient state-of-the-art system features such as sockets, RPC, and other inter-process communication facilities so as to ensure reliable communications.
- a set of specialized *portmapper agents*, distributed on each site, that will manage the low-level aspects of the communication (registering agents, remembering ports, creating UDP or TCP connections) and provide basic mechanisms such as forwarding and broadcasting.

## 4.3 Agent programming

The art of agent programming is to design and implement the different components of an agent, and to assemble the pieces of the puzzle. It is to be noted that some of these elements, although defined at design time, may evolve at runtime.

From a programmer' s viewpoint, an agent is composed of:

- an *interface*, used to communicate with the other agents. This part is automatically generated from the compilations of the protocol specifications;
- a *body of code*, that implements the agent' s basic skills, i.e. the set of automatized operations that an agent is able to perform: actions, perceptions, computations, inferences, etc. This part is usually composed by a set of pieces of code (programs, object files), written in a conventional programming language and available in a compiled form;
- a *controller*, that take decisions, send and receive messages, and monitors the body of code.

For the realization of the controller part, we have designed a special agent-oriented language whose main features are described below:

The language is based on a C/C++ interpreter and offers all the power of a good classical programming language, namely simple types (string, integer, float, boolean, enumerate), type constructors (array, list, structure, union, pointer), and control structures (sequence, branch, loop, call). Some syntactical facilities are also provided, such as the ability to define operators, to overload function definitions, or to use macros (thanks to the C preprocessor).

Interfacing to the C/C++ world is then extremely easy and efficient, due to the syntactical and semantic proximity of the languages. For instance, one can write things like:

```
extern ''C++'' {
#include <some_header.H>
}
```

and use directly the types or the functions defined in the header. Even more interestingly, it is possible to load an object code dynamically "on the fly". It is then easy to design wrappers around existing applications or to write heavily-used functions in a conventional language, and to link these pieces of code either at design time or at runtime.

Moreover, the language has been extended so as to provide a fully object-oriented language, with a meta-object protocol directly inspired from CLOS, including: objects, classes (which are themselves also objects), class inheritance, generic functions, methods, and daemons. At runtime, objects are defined by an interface part and by an implementation part (a pointer to a data structure).

Finally, the language, which is interpreted, offers meta-level features such as: meta types (type, class, procedure, expression), quotes and backquotes (e.g. the expression *"x=1+2*3"*) and a set of built-in functions (parse, eval, load, declare, define, etc.). In particular, any expression can have a textual form (a character string that can be parsed), an interpreted form (a tree-based structure that can be evaluated), or a compiled form (a piece of object code that can be executed). It is therefore possible for an application to represent, reason about, and even to change its own code, e.g. by building a new procedure from scratch and by compiling it.

## 5 Application: an architecture for collaborative problem-solving

Let us consider a composite system composed of one or several human agents, various pieces of software, and a set of devices (screens, robots, whatever). Let us suppose that we want these agents (human or software) to accomplish a set of tasks requiring some kind of collaboration between them.

Let us first try to analyse the nature of that collaboration. Collaborating means namely contributing to the activity of a group: we can then view the collaborative activity as the result of a sum of individual contributions. Moreover, for the purpose of this work, we will assume that such contributions can be categorized given their rhetorical nature (proposition, critic, explication, evaluation, decision, application) and the level at which they occur. We have identified the following levels:

**data:** basic informations, observations;

**interpretation:** hypotheses, models, viewpoints on the situation;

**orientation:** objectives, high-level strategies;

**commitment:** intentions, contracts, tasks;

**organization:** plans, task allocation;

**results:** products of a task, partial solutions.

Consequently, if the structure of a collective activity is compatible with this model, then we can define a set of elementary collaboration acts, or operators, characterized by: an agent, an object (e.g. an hypothesis), a level (e.g. interpretation) and a type (e.g. proposition).

Using the previous platform, we have therefore designed a flexible, multi-level architecture for collaborative problem-solving. Each level is defined by:

- A *working memory*, representing the current state of the collective activity, and which is intended to become the real "living heart" of the community. The role of such a structure is to become an open interaction space where agents can express general-purpose opinions, exchange points of view, make propositions, offer and request services, transmit results, etc. At design time, the memory is originally partitioned into different levels (data, situation, objectives, tasks, plans, results) and can contain informations, models, propositions, critics, explanations, decisions, etc.

- A *knowledge base* collecting information about the domain, the group, or the way to solve problems: schemes, ontologies, task models, methods, procedures, conventions, policies, etc. These data can be used e.g. by classical knowledge-based systems to infer new information, generate hypotheses, propose a plan of actions, etc.

- A set of man-machine or software *interfaces*, giving agent-friendly access to the shared memory and offering a set of basic actions such as: visualize, query, criticize, explain, inform, request, command, etc. Ideally, these interfaces should be customizable to agent's needs and offer facilities like data filtering (present only informations that match agent's foci) and dog watching (trigger some action, e.g. inform the agent, whenever some condition occurs).

- A set of public-spirited *facilitators*, i.e. specialized software agents devoted to the well-being of the community in general and to the cooperation of its members in particular. The set of facilitators includes individual assistants (attached to a specific user) and inter-agent coordinators (acting as mediators between two or several agents).

Using the range of facilities described in the previous section, users can decide for instance to change the structure of the shared memory, to add facilitators, or to integrate new agents into the framework. Therefore, thanks to the flexibility of the underlying platform, this framework can be considered as an *open system* [Hewitt and de Jong, 1983].

Such an architecture can be used, for instance, to support various coordination strategies used by human beings, such as iterative model refinement [Bond and Ricci, 1992].

## 6 Conclusion

We claim that flexibility and openness are major requirements for the usability of a cooperation framework. The present approach emphasizes this need for flexibility and paves the way for a general approach to the design of collaborative systems. In particular, our multi-agent platform has interesting properties, such as:

**distribution.** Agents can be physically distributed on different machines, or run as different processes on the same machine.

**genericity.** The approach is independent of the application domain or of specific implementation choices;

**flexibility.** Different modes of communication and coordination strategies can be implemented;

**inter-operability.** Heteregoneous agents and existing applications can easily be integrated within a common framework;

**reactiveness.** Agents do not follow fixed plans. They can suggest, inform, criticize, explain, negotiate, etc. in response to the actions of the others;

**self-organizability.** Roles and tasks can be dynamically allocated by agent themseves. Real collaboration between agents requires a joint effort to define the organization of their collective activity.

**efficiency.** Communication protocols are reliable and based on efficient inter-process communication mechanisms. Most of the code is written in C-based languages, and can be compiled for maximum speed.

However, our current implementation is still a prototype. What is needeed is to test such a framework on real applications, especially in the industrial world.

We are conducting preliminary experiments in the domain of car collision analysis, for the design of a system where experts of different specialties (psychology, physics, road infrastructure) collaborate in order to build simulation scenarios and understand the nature of collisions. We hope that this experience will give us better understanding of the impact of such an approach on the design of collaborative systems.

### Related work

This work is to be related to recent researchs that aim at applying multi-agent techniques to the design of generic cooperation architectures ([Gasser, 1993]), especially ARCHON [Cockburn and Jennings, 1995] and SHADE [McGuire and al., 1993].

A few approaches have also tried to merge the techniques of distributed problem solving (DPS) and multi-agent systems (MAS). For instance, the TEAM architecture [Lander, 1994] is composed of 1) a common memory that represent the current state in problem solving (a blackboard portioned into: solutions, problem specifications, coordination strategies, messages), 2) a framework manager that manages the common memory and the communication with the agents, and 3) an agent set that is connected to the framework. However, the current prototype does not allow to distribute agents into separate processes.

### Acknowledgments

This work was partly sponsored by the European Community under the scope of the ESPRIT working group MODELAGE (contract EP:8319) [Shobbens and al., 1994]. Comments from Olivier Corby and other members of the ACACIA team have helped to shape the viewpoints expressed in this paper.

## References

[Bond and Ricci, 1992] Bond, A. and Ricci, R. (1992). Cooperation in aircraft design. *Research in Engineering Design*, 4(2):115–130.

[Bond and Gasser, 1988] Bond, A. H. and Gasser, L. (1988). *Readings in Distributed Artificial Intelligence.* Morgan Kaufman Inc.

[Cockburn and Jennings, 1995] Cockburn, D. and Jennings, N. R. (1995). *ARCHON: A Distributed Artificial Intelligence System for Industrial Applications.* Morgan Kaufman Inc., Wiley.

[Gasser, 1993] Gasser, L. (1993). *DAI approaches to Coordination*, pages 31–52. Kluwer Academic, Dordrecht.

[Hewitt and de Jong, 1983] Hewitt, C. and de Jong, P. (1983). Analyzing the roles of descriptions and actions in open systems. Technical Report AI Memo 727, MIT.

[Lander, 1994] Lander, S. E. (1994). *Distributed Search and Conflict Management Among Reusable Heterogeneous Agents.* PhD thesis, U. of Massachusetts Amhert. available as CS technical report 94-32.

[Marick and al., 1994] Marick, V. and al. (1994). A distributed system for CIM. In *Proceedings of DEXA '94*, Athens, Greece.

[McGuire and al., 1993] McGuire, J. and al. (1993). SHADE : Technology for Knowledge-Based Collaborative Engineering. *KSL WWW Server.*

[Shobbens and al., 1994] Shobbens and al. (1994). *ModelAge, A Common Formal Model of Cooperating Intelligent Agents.* Kluwer Academic. B.R.W.G. proposal.