ARCHON: Theory and Practice

N.R.JENNINGS
Dept. Electronic Engineering
Queen Mary & Westfield College
Mile End Road
London E1 4NS
UK
nickj@qmw.ac.uk

T.WITTIG
Atlas Elektronik GmbH
Research Department
Sebaldsbrucker Heerstr. 235
W-2800 Bremen
Germany
T_Wittig@eurokom.ie

ABSTRACT. This paper describes the state of the ARCHON project at its halfway stage. It outlines why a Distributed AI approach is appropriate for large industrial applications, the benefits which accrue and highlights those domain characteristics which have had significant bearing on the design process. The ARCHON functional architecture is described and a clear mapping with the design forces is made. The problem of constructing a general architecture which is sufficiently powerful for real-size applications is raised and two mechanisms used within ARCHON for achieving this objective are identified. The first mechanism is to combine generic, declarative knowledge related to cooperation and control with situated action formalisms which are tailored to the particular application. The second approach is to provide general structures which can be instantiated with domain specific information.

1. INTRODUCTION

This paper provides an introduction and overview of the ongoing ESPRIT II Project P-2256 ARCHON (Architecture for Cooperative Heterogeneous On-line Systems). The aim of the project is to construct a general purpose multi-agent system architecture for industrial applications and the work described herein describes the state of play at the halfway stage. This work was motivated by the observation that computer-based industrial applications are currently testing the limits of system engineering in terms of their sheer size and amount of knowledge they need to embody. Monolithic solutions are becoming unwieldy and difficult to engineer into reliable consistent products.

To overcome this complexity barrier, there has been a tendency towards modular design over the past few years which has resulted in functionally distributed systems. These smaller and more manageable components are usually ordered in some hierarchical fashion with clear, predefined communication links. Although such design increases the maintainability of a complex system, as well as its adaptability to evolving requirements during the lifecycle, it usually keeps the overall control at a central location. From here, all coordination between components is handled and the more complex such components become (like expert systems, for example) the greater the concomitant control bottleneck. For example, in large industrial applications, which consist of a number of Supervision and Control (S&C) systems for specific sections of the overall process, the activation of tasks in the subsystems and the decision of what data to exchange between them depends on the situation or the state of the entire process. In a centrally controlled system, this assessment can only take place at the central location, taking into account the different views of all S&C systems involved and hence leads to severe delays.

To alleviate this control bottleneck and increase the flexibility of coordination (data exchange, task activation) among the subsystems, the next step in system design is the decentralization of control. Such design not only allocates much more responsibility to the subsystems but also requires them to coordinate their tasks. This additional coordination or cooperation functionality is the focus of the Distributed AI (DAI) approach of ARCHON.

Section two outlines the need for a DAI approach to industrial applications, citing the benefits and providing key domain characteristics which need to be reflected in the framework's design. Section three details the role of a cooperation framework in industrial applications, eliciting the requirements and showing how these lead to the ARCHON functional architecture. Finally section four places ARCHON within the context of existing DAI research and highlights its niche. It describes the principles which underlie the design of a framework which has to be applicable to a wide class of problems, yet be sufficiently powerful to operate in highly complex, time constrained environments.

2. Why Distributed AI is Needed For Industrial Applications

2.1 BENEFITS OF DISTRIBUTED AI IN INDUSTRIAL APPLICATIONS

The general benefits associated with using a DAI approach stem from the distribution of control and data and the increased software modularisation which can be achieved [1], [2], [3]:

- Modularity: traditional advantages from software engineering; including decreased component complexity and greater clarity
- Speed: sub-systems can operate in parallel
- Reliability: system can continue even if part of it fails
- Knowledge acquisition: easier to find experts in narrow domain
- Reusability: small independent system could be part of many cooperating communities

In addition to the above, within the domain of industrial systems, the benefits of enhanced problem solving, easing the burden of the system operator and increased reliability are especially pertinent [4]. Individual problem solving can be enhanced in quality as well as speed by sharing information and processing in an efficient manner between community members. If an operator is in charge of several systems working on the same process then he has to manually pass information between them and ensure their activities are coordinated; however by automating this interchange, the operator can concentrate on the higher level (cognitive) actions for which he is better suited. Finally reliability is enhanced because if part of the system fails, then performance should degrade in a graceful manner because both the control and data is distributed.

2.2 DESIDERATA FOR INDUSTRIAL SYSTEMS

When considering a DAI approach to solve industrial problems, several important characteristics of the domain need to be considered:

- there is a substantial amount of preexisting software
- industrial systems are complex and require many diverse types of activity to be performed
- the operator is an integral component of the problem solving community

Typically within the domain of industrial systems, software has been developed in an ad hoc fashion when the company has perceived that certain functions could profitably be automated using available technology. The result is that such companies posses a large number of standalone systems, developed at different times, by different groups of people and utilising different problem solving techniques. These systems all operate within the same physical environment (i.e. take input from and produce output to the same process) and could benefit from interaction (sharing of information and problem solving expertise) with other such standalone systems. However since they were not been constructed with the aim of integration in mind, they employ techniques and representations which are best suited to the particular problem(s) they were designed for - ensuring that data/knowledge formalisms and the associated reasoning techniques are specific to that particular implementation. Faced with this situation, there are several courses of action open to the system designer:

- rebuild all the existing software in a common environment, so that they all share common representations, reasoning mechanisms and knowledge semantics
- construct a framework into which the existing systems can be incorporated (with minimal modifications) and allow them to interact with each other.

  Typical examples of the modifications which have to be made include allowing the cooperation framework to affect the control instance of the system and altering the system's presentation of information to the user (to incorporate the fact that the system is now a team member, not merely an individual).

In most cases, the latter option is preferable because of its lower cost, resource usage and general risk. However this means that problems associated with heterogeneity become an important issue and require attention within the agent architecture. There are many different levels of heterogeneity which need to be dealt with, ranging from different operating systems and programming languages to issues associated with the inherent distribution of tasks within the community and the overlap in the understanding of domain concepts between agents [5].

Large S&C systems are also inherently heterogeneous with respect to the various subsystems involved. Subsystems are targeted to specific and quite diverse tasks [6]:

Diagnosis: deliver an understanding of world state given some information about this world

Planning: sequence a set of possible actions

Control: particular case of planning where actions are executable and low-level

Supervision: reflecting decision link between diagnosis of a dynamic system and the alternative actions needed for handling exceptional situations

The wide range of problem solving techniques necessary for each of these sub-areas means that a common technique and knowledge/data representation would be infeasible and hence issues related to heterogeneity are important even when constructing systems from scratch. To obtain the necessary complexity, different techniques will be needed for different tasks and even within a single generic task more than one technique may be employed (eg in diagnosis there may be a heuristic-based component and a model based component). For example, an alarm analysis system in an electricity distribution network has to cope with large numbers of incoming events and map these to the topology of the network held in huge, real-time databases. In contrast to this, an electricity load management system that plans and controls the overall load of a network, only deals with a small number of real-time events but instead has to cope with non-monotonic planning based on crude qualitative models of the consumers in the network. So it is not surprising that these systems not only differ widely in their software structure but often run on quite different hardware platforms. Although the current trend in S&C systems in industrial applications is towards more homogeneous and open systems, the coordination problem is at best just shifted a level higher. Furthermore, other applications such as ship handling and control [7] will remain heterogeneous. This need to deal with issues associated with heterogeneity contrasts with much of the early DAI work in which agents had homogeneous structures (e.g. BEINGS [8] or DVMT [9]), were capable of carrying out identical tasks (eg Air Traffic Control [10]) or were assumed to have identical domains of discourse.

Due to the critical nature and potential risks associated with industrial systems, it is inevitable that human operators will remain an integral component for the foreseeable future [11]. Therefore when designing a community of cooperating agents it is essential to ensure the operator is included as an active problem solving member, able to volunteer information, carry out problem solving tasks, focus activity and so on. Some of the issues which need to be addressed include the allocation of tasks between operators and the artificial agents, how to design features of the interaction so that the best capabilities of operator and computational agents are utilised, how authority can be implemented and so on. These issues are discussed more thoroughly in [12], [13].

3. ARCHON Functional Architecture

3.1 THE APPLICATION CONTEXT

In order to describe the required functionality of the ARCHON approach, we will use the ‘idealised operator analogy’: We assume a set of independent S&C systems, each controlled by an operator with the only coordination taking place amongst the operators (depicted in the left part of Figure 1). The operators share some overall goal, e.g. the economic efficiency of a power distribution network or the safe operation of a large vessel. Each operator will derive his personal aim for controlling his S&C system from such an overall goal. Naturally he is aware that he is not working in isolation: he is aware of the other operators and has some understanding of what their tasks are and what they can achieve. This knowledge enables him to either ask for help or to respond to requests from them.

This operator level is the target of ARCHON and the problem is to identify the functionality a system requires in order to act on this level. The right part of the figure lists the main requirements for an architecture that in the end will achieve the cooperative link between independent systems.

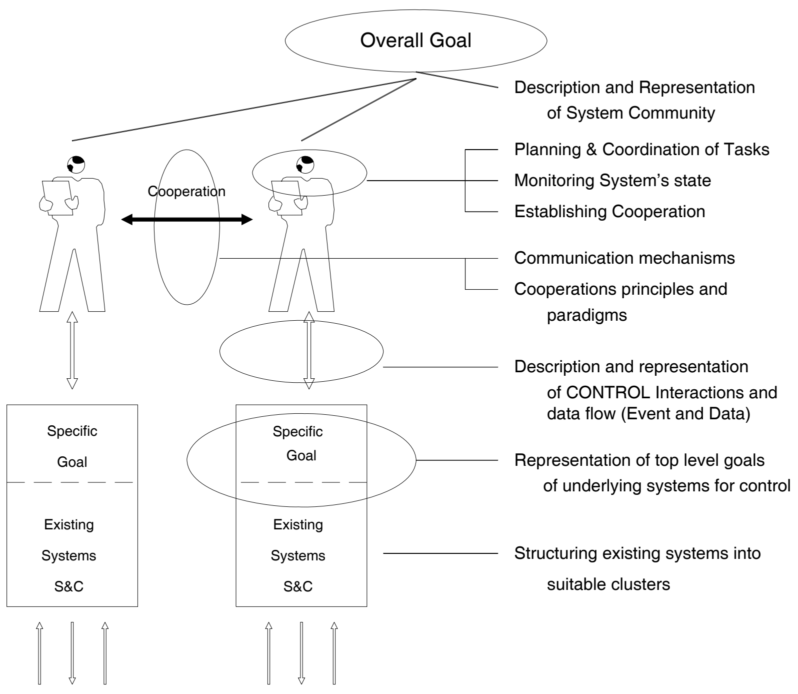
Overall Goal
Description and Representation of System Community
Cooperation
Planning & Coordination of Tasks
Monitoring System's state
Establishing Cooperation
Communication mechanisms
Cooperations principles and paradigms
Description and representation of CONTROL Interactions and data flow (Event and Data)
Specific Goal
Existing Systems S&C
Specific Goal
Existing Systems S&C
Representation of top level goals of underlying systems for control
Structuring existing systems into suitable clusters

Figure 1: Requirements for a Cooperative Framework

We will go briefly through these points and then indicate how they are mapped into the functional architecture of ARCHON.

3.2. THE REQUIREMENTS

The overall goal

The general goal of the application has to be represented somewhere in a suitable form. The heterogeneity of the application suggests a more loosely coupled approach, implying multi-agent system techniques rather than a distributed problem solving ones. Consequently, we are not representing a 'Common Goal' anywhere in our architecture but only goals of the agents which, taken together, form the overall goal of the community. This is one of the most important tasks the designer of such a cooperative system has to pay attention to and is very closely related to the way the community is represented within each agent. No agent has to model all the others, only those it is likely to interact with. By representing skills, interests and goals of its acquaintances an agent is able to specifically involve others in its own problem solving objective and at the same time 'respect' their autonomy [4], [14].

The view on the own system

Just as an operator works on his S&C system, the ARCHON framework functionality has to provide this view. This involves monitoring and assessing the system's state on the one hand and providing the means to interact and control it on the other. To do this, the control interactions for the domain system have to be described and represented in a suitable way. In ARCHON this is primarily done through an event-based mechanism that allows time and data to be captured. As described in more detail in section 4, there are two different ways to respond to events from the domain-system: reactive and reflective. The first one relates to ‘standard’ responses in nominal situations, i.e. when the system or the application is behaving as expected. Reactive responses do not require explicit reasoning and can be dealt with by pre-compiled plans. In exceptional situations, however, such pre-compiled plans would fail. Much like an operator has to ‘think’ about such problems, ARCHON has to provide means for explicit reasoning, i.e. it has to contain a reflective component.

The operator usually has a certain amount of freedom in assigning tasks to his S&C system, based on his knowledge of the system’s tasks and goals. In order to flexibly control the system through ARCHON, its top level goals and accessible tasks have to be represented. Again, an ARCHON system designer has to pay attention to this point: As long as the domain system does not permit different control options there is not much point in building a cooperative framework on top of it. It may well be that the designer has to cluster existing systems into suitable bundles to achieve this flexibility of control.

The view on the community

Based on the assessment of the state of the domain system, the need for help from other agents may arise. Equally, preempting urgent requests from other community members, an operator may volunteer certain information derived from his system to others. Through the representation of skills, interests and goals of other agents, coupled with the appropriate reasoning facilities, ARCHON provides this functionality. Being the central point of a cooperative framework on top of existing systems, this functionality should be as independent from any domain systems as possible. In fact, as described later, the current architecture prototype provides a number of such independent rules for detecting the need for cooperation and for determining the best way of establishing it.

3.3 GENERAL ARCHITECTURE

We will explain the architecture of ARCHON in two steps: firstly, the general modules that can be derived from the requirements identified in the previous section will be presented and, secondly, it will be shown how these modules map with the ‘operator analogy’.

Figure 2 sketches the modules of the ARCHON layer and shows the interface to the domain level system. This system is called ‘Intelligent System’ mainly because of historical reasons. In the early days of the project the ARCHON Framework was seen more as yielding a community of cooperating expert systems, i.e. knowledge based domain systems. Without excluding this possibility, we have now a broader view on the functionality of domain systems, allowing information processing systems in general (provided they allow sufficiently flexible control as described above). In fact, we distinguish two levels of ‘intelligence’ in an ARCHON system: On the domain level and, more important in this context, on the coordination and cooperation level. Each ARCHON layer is in itself a knowledge based system reasoning about its domain system and the coordination within the community, but not solving any of the domain problems.

The architecture needs a communication facility, which is called the High Level Communication Module (HLCM). This module is actually a layer, providing services to the other components of the

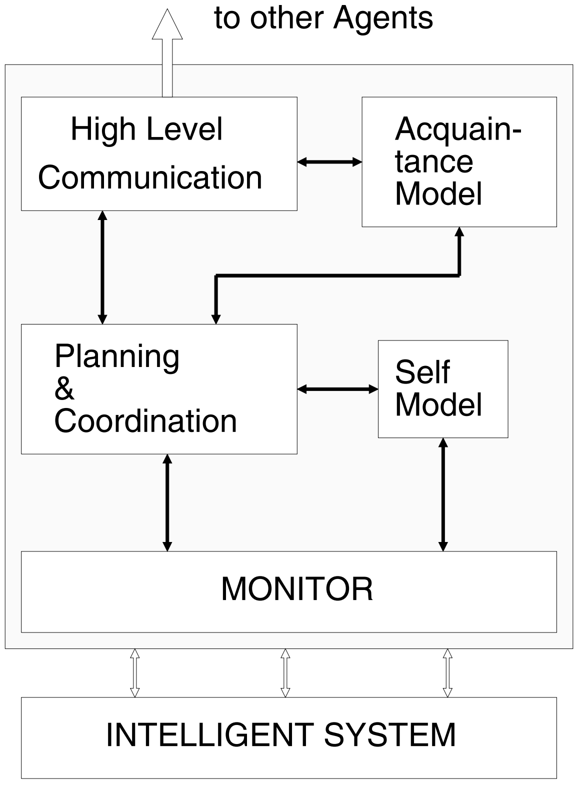

Figure 2: Simplified Agent Architecture

ARCHON layer. It is called High Level since it not only provides the communication facilities (achieved through a Session Layer implementation) but also services such as intelligent addressing and filtering. For example, if the domain system produces a result that may be relevant for other agents, the Planning and Coordination Module just asks the HLCM to send it to all interested agents without specifying them.

The Agent Acquaintance Model (AAM) contains appropriate models of other agents in the community in terms of their skills, interests, current status of workload etc. and is discussed more extensively in [14]. These models are prerequisites for any coordination of activities among agents such as asking for help. The intelligent addressing facilities of the HLCM make use of this model to find the agents interested in a specific result.

Much like the AAM models other agents, the Self Model (SM) is an abstract representation of an agent's domain system. It primarily contains information about the current state of this system, i.e. its workload, or what tasks are being executed but also embodies the pre-compiled plans (behaviours) for the reactive part of the control. These plans are accessed by the Monitor which is responsible for the control of the intelligent system and for the passing of information to and from it.

The Planning and Coordination Module (PCM) represents the main reflective part of the ARCHON layer. If an exception occurs it is the task of the PCM to reason about it and find a way out. Its influence on the Monitor is mainly through the behaviours stored in the SM and is achieved by

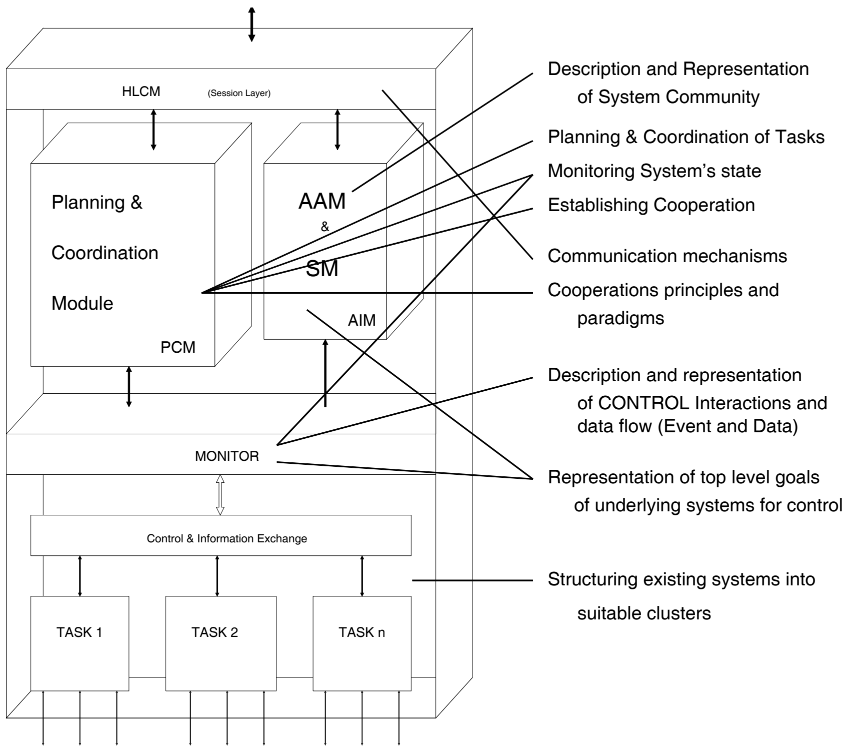
HLCM
(Session Layer)
Planning &
Coordination
Module
PCM
AAM
&
SM
AIM
MONITOR
Control & Information Exchange
TASK 1
TASK 2
TASK n
Description and Representation of System Community
Planning & Coordination of Tasks
Monitoring System's state
Establishing Cooperation
Communication mechanisms
Cooperations principles and paradigms
Description and representation of CONTROL Interactions and data flow (Event and Data)
Representation of top level goals of underlying systems for control
Structuring existing systems into suitable clusters

Figure 3: Requirements mapped into the architecture

changing appropriate parameters and activation information. It also contains some reactive parts but only with respect to cooperation initiation and response. For example, having detected the need for cooperation and having determined what to ask from another agent, the selection of the appropriate cooperation protocol may simply be a function of the current workload of the domain system and not require any further reasoning. The same may be true when receiving a request for cooperation from another agent.

3.4 MAPPING OF THE REQUIREMENTS TO THE ARCHON ARCHITECECTURE

Figure 3 shows the correspondence of the architecture modules to the previously identified requirements. This figure is self explanatory, except for one functionality not mentioned before: the Agent Information Management (AIM). We have explained that the Agent Models contain all relevant information about the agents. This refers not only to static information such as skills, but also to the data to be exchanged. As long as the amount of such data is small, this approach is acceptable. But in applications such as electricity management we can find easily data-sets of several Mbytes in size, e.g. the updated topology of a section of the network containing all elements with their current status. A number of agents may be interested in certain aspects of such data, but rarely in all of it. So either the agent where this information was generated sends the complete set to all interested agents so that they can select whatever they want and discard the rest or the agent could select all subsets that may be required by some other agents. Both approaches are unsatisfactory; the first because it would overload the communication channels with irrelevant information, the latter because it would consume considerable computation resources for results that may not be needed in the end. Thus, the solution developed for ARCHON consists of a distributed database in which all the agent models are implemented. This distributed DB is embedded in the general communication facilities of ARCHON, but the communication is hidden from the other modules. The main idea in this approach is that any information generated is stored and kept at the source, i.e. in the self model of the generating agent. A set of global accesssor functions are available enabling information stored at other agents to be obtained and selected [15].

4. Achieving Power and Generality

4.1 RELATIONSHIPS TO OTHER SYSTEMS

Within the field of DAI, two main types of software tool have been developed: *integrative systems* and *experimental testbeds* [1]. Integrative systems, such as ARCHON, provide a framework to combine a variety of problem specific tools and methods into a useful whole, whereas experimental testbeds have their main emphasis on controlled experiments in which both measurement and monitoring of problem solving activity are the prime consideration. Within the former category two broad classes of system can be identified:

- systems which test a specific type of problem solver, a specific coordination technique or a particular domain

  eg DVMT [16], ATC [10], HEARSAY II [17], Pilot's Associate [18]

- systems which are *general* to some extent

  eg MACE [19], ABE [20], CooperA [21]

By their very nature, these two types of system have been built to illustrate different concepts and have been applied to different types of problem. The specific systems are usually built as solutions to particular problems (eg vehicle monitoring, air traffic control) which may be fairly complex but they are often difficult to generalise. General systems and architectures, on the other hand, are built to show that a particular problem solving architecture is applicable for solving several different kinds of problem, but are often applied to toy problems (eg N-Queens, Blocks World) or gross simplifications of some real world problem and often do not scale up. This means that, in general, the specific systems are difficult to generalise and the general systems do not have the necessary power to solve real size industrial applications. One of the key objectives of the ARCHON system is to bridge this gap - providing a system which is both applicable to a wide class of applications and yet powerful enough for real size problems. Ideally such a system would handle all applications and any problem decomposition of that application; however to limit the scope several restrictions have been imposed. Firstly the domain will initially be limited to industrial applications and secondly the types of intelligent system which will be incorporated are assumed to be coarse grained and loosely coupled. These characteristics fit well with the desire to incorporate preexisting systems: such systems are capable of significant processing and can solve most of their problems by themselves (since they were conceived and built as standalone).

Related to the power-generality dimension, is the abstraction level of the multi-agent framework: this can range from a programming language (eg ACTOR languages [22]) to a system in which all the developer has to do is fill in the blanks (a *skeletal system* [20]). A programming language can be used to solve most problems which the designer is likely to face (i.e. it has the necessary power); but requires significant effort (especially for complex problems) to construct the system since all the concepts necessary for social interaction need to be coded from scratch. A skeletal system, on the other hand, greatly reduces the developer's effort since much of the structure already exists; its drawback being that the structure may not be suitable for a wide variety of applications and it may be difficult to fine-tune the system to obtain the desired performance characteristics (i.e. neither general nor powerful enough). Therefore when designing a multi-agent system it is important these considerations are taken into account: ARCHON offers a general framework (the functional architecture) which contains generic knowledge related to control and cooperation (a significant proportion of a skeletal system) and also provides a language (ALAN [23]) in which domain dependent aspects can be embodied. This mixture of abstraction paradigms allows general and powerful systems to be embodied, but possess sufficient structure to ease the application builder's task.

4.2 A HYBRID APPROACH

In order to produce a general and powerful architecture it is important to embody principles which produce systems with the desired characteristics. Unfortunately there are no mechanisms or techniques which are universally appropriate: it is possible to identify some techniques which increase the power and some which enhance the generality. In order to construct a system which exhibits both sets of characteristics, a hybrid approach was adopted - mixing techniques which enhance power with those which improve generality. Techniques for enhancing power include: adapted from [24]

- Specialisation Into Modalities

  By identifying functions which need to be performed and developing specialised subsystems to implement them, the overall system performance can be enhanced and a clear modular architecture produced. When designing a cooperation framework the distinct functions are related to controlling the underlying system, to assessing the situation (both locally and globally) and for acting in a social environment.

- Compilation of knowledge and behaviour (Reactive Mechanisms)

  By compiling down knowledge and reasoning mechanisms, processing power can be greatly improved. Rather than having to reason about which actions should be taken an agent merely has to recognise the situation and carry out the associated action sequence.

In order to enhance generality, the following strategies can be applied: adapted from [24]:

- Explicit Representations

  Explicit representations of the world and an agent's actions allow high level descriptions of agent behaviour to be formulated. Such mechanisms enable it to operate in a wide variety of situations as it can embody general principles rather than be tightly coupled to the specific situation (cf reactive mechanisms).

- Generic Structures

  By using structures which are meaningful in many different situations to represent domain specific information, the associated reasoning mechanisms can obtain a degree of generality. That is, base the behaviour on the structure imposed by the generic representation mechanism rather than the idiosyncracies of the problem being tackled.

Adherence to these principles can be identified within the ARCHON functional architecture and its associated knowledge and inferencing schemes. Specialisation into modalities can be observed within the functional architecture: the PCM is responsible for assessing the global situation and for dealing with issues involving other agents, the monitor for dealing with local control activities, the HLCM with communication and so on. The other points were also used to guide architectural decisions: the knowledge representation scheme is a combination of a declarative, rule based approach (explicit representation) and of a situated action component (compiled knowledge and behaviour). The functional architecture also has several generic structures and mechanisms which need to be instantiated with domain specific information (i.e. the blanks in the skeletal system are filled in). These last two issues are developed further in the following subsections.

4.2.1 Pre-Compiled Behaviours & Reflective Components.

In situated action systems: agents react to situations and do not reason about the world, both agents and actions are simple in nature and global properties are seen as an emerging from the interaction of behaviours [25], [26], [27]. Their advantage is that because agents neither maintain nor reason about the world in an explicit manner (all the reasoning is carried out by the designer and is pre-compiled), they are very fast and can respond rapidly to changing environmental conditions (i.e. they are powerful). However as they are tightly coupled to the environment their behaviour is domain dependent and inflexible in that they can only respond to events for which they have a predefined stimulus-response pair. Alternatively systems which do maintain an explicit representation of the world and encode knowledge in a declarative manner typically have the opposite properties - they can be made more general and decoupled from the environment, but often lack the speed to handle complex reasoning tasks in unconstrained domains.

In order to construct a general-powerful system the desirable properties of both forms of knowledge representation need to be utilised by appropriately combining the reflective and reactive components. Three main functionalities which need to be supported by the ARCHON architecture have been identified:

- Local Control (Monitor)

  Requires a fast response to numerous events and situations arising in the domain system. Such control will vary from application to application and is difficult to generalise, therefore it is well suited to being encoded in a situated action formalism.

- Social Activities (PCM)

  The number of cooperative requests occurring will be relatively few in number because agents will spend most of their time engaged in problem solving rather than in communicating information and are typically capable of solving a substantial proportion of their problems themselves. It is also fairly easy to identify the types of cooperation request which may occur and the reasons behind them. For example agents may request pieces of information or ask for tasks to be performed because they cannot be realised locally or because it is deemed better to ask for somebody else to carry them out. As such activities can be described at a relatively high (general) level and because they are relatively few in number, such knowledge is well suited to a reflective reasoning mechanism.

- Situation Assessment (PCM and Monitor)

  Assessment activities are intermediate in number and in terms of their generality. Some high level assessment functionality can be considered general, whereas other components need to be specialised for a particular application. Therefore in terms of knowledge representation, this functionality has been implemented as a mixture of generic rules and reactive mechanisms.

To provide an example of how reflective and reactive components can be combined, consider the following example taken from the PCM. One of the tasks of the PCM is to decide upon a particular cooperation protocol once the need for social interaction has been detected. At present, two cooperation protocols are embodied: *client-server* and *contract net* [28]. In the client server model, the agent initiating the cooperation decides with whom the interaction ought to be (based on information located in its acquaintance models) and sends a cooperation request directly to that agent. This protocol has the advantages of a low communication overhead and that the interaction can be started immediately. However it has the disadvantage that the intended server may turn out to be unsuitable; for example it may be just about to embark on a time consuming, high priority task - ensuring the request will not be answered in a timely fashion. The contract net is more protracted: the agent initiating the social activity sends out a request to all its acquaintances, waits for bids from each of them which it evaluates before deciding which should be awarded the activity. This protocol has a higher communication overhead and takes a longer time for the interaction to be established, however it has the advantage that the agent eventually chosen is likely to be better suited to tackling the problem (since its bid states the time by which the activity should be completed). These properties can be encoded in general, declarative structures:

```
(rule select-protocol1
    (IF (COMMUNICATION-RESOURCES-OVERBURDENED ?Agent))
    (THEN (ESTABLISH CLIENT-SERVER-COOPERATION-INSTANCE ?Agent ?Task)))

(rule select-protocol2
    (IF (AND (IMPORTANT ?Agent ?Task) (FIXED-DEADLINE ?Agent ?Task)))
    (THEN (ESTABLISH CONTRACT-NET-COOPERATION-INSTANCE ?Agent ?Task)))
```

The first rule states that if communication resources are currently being used to near capacity, then the client server protocol should be chosen because of its inherently lower communication cost. The second one states that if a task is considered important and has a deadline which must be adhered to (but which is not too close) then the most appropriate protocol is the contract net - since it provides a more reliable completion time estimate. It is also possible to have protocol selection based on meta-level considerations:

```
(rule select-protocol3
    (IF (FAILED (ESTABLISH CLIENT-SERVER-COOPERATION ?Agent ?Task)))
    (THEN (ESTABLISH CONTRACT-NET-COOPERATION-INSTANCE ?Agent ?Task)))
```

stating that if client server was selected initially, but that its execution failed (eg the intended server was too busy to respond) then a contract net ought to be established. The rationale behind this being that the contract net reaches a greater number of agents and hence it is more likely that a suitable acquaintance can be found.

Rules 1, 2 and 3 are general in that they can be used in any application in which the client server and contract net protocols are available. It would also be possible to encode the protocol control mechanism using similar general rules since, by their very nature, they follow a set ordering of steps. So in the contract net, with a task announcement the social activity originator has to decide the agents to which the request should be sent, give the contract a unique identifier and then actually broadcast the announcement. It then has to wait until it receives all the bids before deciding to whom the contract should be awarded and then informing the chosen agent of its successful bid. However because of the fairly static sequence of actions and the lack of associated reasoning, it was decided that the protocol could be more efficiently encoded by a pre-compiled behaviour [23]; leaving the reflective component free for more sophisticated reasoning:

```
TASK-ANNOUNCEMENT (TASK REP) {
    TASK-DESCRIPTION (TASK) ((?AGENTS ?TIME ?CANDIDATES))
    CONTRACT-ID () (?CONTRACT-ID)
    BROADCAST(CANDIDATES ‘ANNOUNCE CONTRACT-ID TASK)
    GET-BID(CONTRACT-ID AGENTS TIME) (?LIST-TO-AWARD)
    BRAODACAST (LIST-TO-AWARD CONTRACT-ID TASK)}1
```

This behaviour differs from most of the others in that it is domain independent and it can therefore be used in other ARCHON applications in which the contract net is desirable.

4.2.2 Domain Independent Structures & Domain Dependent Instantiations

There are several mechanisms and structures within the ARCHON framework which can be viewed as domain independent (eg the communication facilities, the acquaintance modelling and the information modelling). This section concentrates on the agent models to illustrate how meaningful behaviour can be obtained from the *structure* of a generic component.

The first example shows how a PCM rule uses the acquaintance model to initiate the social action of spontaneously sending unrequested information. Agent A’s model of acquaintance B contains the slot:

1. This is a sample behaviour written in ALAN [23]. Underlining signifies reference to other behaviours

Slot-Name: Interests

Slot Structure: INTERESTED-IN { <Name_1, Cond_1> <Name_2, Cond_2>,................}

meaning that if A generates a piece of information which appears as the first element of a tuple in the list (eg Name_1,Name_2, etc.) then this information is of interest to B if the expression in the second element of the corresponding tuple evaluates to true. The interpretation of this structure can be made using a generic rule:

```
(rule generate-unreq-info1
    (IF (AND    (HAS-AVAILABLE ?Agent ?Info)
                (ACQUAINTANCE-INTERESTED-IN-INFO ?Agent ?Info ?Acq)))
    (THEN (SEND ?Agent ?Acq Unrequested-Information ?Info)))
```

which states that if information becomes available to A and an acquaintance is interested in it (corresponding condition is true) then the information ought to be sent from A to B as unrequested information.

Whenever tasks are started, this fact is recorded by an agent in its self model component which keeps a record of all active tasks. Associated with each task is the information which has been passed to it, its execution status, priority and the reason for executing it. The rule shown below makes use of this descriptor to obtain the reason why a task and its associated plan was activated. If it was to satisfy an information request, then a generic rule is needed to ensure that the request is honoured if the information is available:

```
(rule finish-social-activity1
    (IF (PLAN-FINISHED ?Plan ?Results (SATISFY-INFO-REQUEST ?Info ?Orig)))
    (THEN  (EXTRACT-DESIRED-VALUE ?Results ?Info ?Desired-Value)
           (ANSWER-INFO-REQUEST ?Info ?Desired-Value ?Orig)
           (DELETE-MOTIVE SELF ?Plan (SATISFY-INFO-REQUEST ?Info ?Orig))
           (SEE-IF-FURTHER-MOTIVES ?Plan ?Results)))
```

Again this rule is providing the semantics of the self model component which tracks an agent's current activity.

5. Conclusions

This paper summarises the efforts of the ARCHON consortium over the past two and a half years. The view presented here is only partial and the interested reader is referred to [29] for further details. We have identified heterogeneity and the ability of an operator to actively participate in problem solving as critical design forces in the construction of cooperation frameworks for industrial applications. The type of problems which ARCHON is addressing have been detailed and the associated requirements have been mapped into the functional architecture. Finally ARCHON's hybrid approach and the use of generic structures have been identified as key techniques for confronting the power-generality problem.

During the remainder of the project we will concentrate on applying these concepts more rigidly to applications in order to eventually arrive at our final goal: the ARCHON framework as an industrial product for a wide range of supervision and control applications. This involves a two step approach: firstly to restructure the control level of already existing applications and at the same time develop the ‘principle ARCHON layer.’ This refers mainly to designing and implementing the Monitor that finally deals with the intelligent systems of the application and, by its very nature, is highly domain dependent. But it may also involve fine-tuning and possibly extending the PCM part of the ARCHON layer. Although a large part of it is domain independent (the generality of the ARCHON approach), some application specific rules for cooperation and situation assessment may be required. Finally this step concerns the design and ‘instantiation’ of the agent models. Once all of this has been achieved, the cooperative behaviour of the agent community can be tested, provided a suitable simulation or provisional interface to the ISs is available.

The second step concerns the porting of this implementation to the real environment. For efficiency reasons and smooth embedding of the ARCHON layer into existing hardware and software environments, it is expected that large parts of our current LISP implementation will be ported into C or C++.

Acknowledgments

The work presented in this paper reflects the efforts of the whole ARCHON consortium whose partners are Atlas Elektronik, JRC ISPRA, Framentec, Labein, QMW, IRIDIA, Iberdrola, EA Technology, Amber, Technical University of Athens, University of Amsterdam, Volmac, Cern and University of Porto.

In particular we would like to acknowledge the contribution of the following individuals: Erick Gaussens (FTC) and Abe Mamdani for general work on the functional architecture, to Jean-Marc Loingtier and Daniel Gureghian (both FTC) for developing the situated action language ALAN and working on behaviours, to Jochen Ehlers (ATLAS) for work on the PCM and Eugenio Oliveira for work on the generic rule concepts.

References

[1] Bond,A.H. & Gasser,L., (1988), “*Readings in Distributed Artificial Intelligence*” Morgan Kaufmann.

[2] Gasser,L., & Huhns,M.N., (1990), “*Distributed Artificial Intelligence Vol. II*”, Pitman.

[3] Huhns,M.N., (1989), “*Distributed Artificial Intelligence*”, Pitman.

[4] Jennings,N.R., (1991), “*Cooperation in Industrial Systems*” Proc. ESPRIT Conference, Brussels.

[5] Roda,C., Jennings,N.R. & Mamdani,E.H., (1991) “*The Impact of Heterogeneity on Cooperating Agents*”, Proc. AAAI Workshop on Cooperation among Heterogeneous Intelligent Systems, Anaheim, Los Angeles.

[6] Gaussens,E.J., (1990), “*Needs and Opportunities for Expert Systems in Process Control*”, Vacation School for Process Control, University of Strathclyde, Scotland.

[7] Wittig,T., (1992) "*Cooperating Expert Systems*", In: Universidad Internacional Menendez Pelayo, Summer-School on Expert Systems at Santander - 91.

[8] Lenat,D.B., (1975), "*BEINGS: Knowledge as Interacting Experts*", Proc IJCAI, pp 126-133.

[9] Lesser,V.R. & Erman,L.D., (1980), "*An Experiment in Distributed Interpretation*", IEEE Transactions on Computers, 29 (12), pp 1144-1163.

[10] Cammarata,S., McArthur,D. & Steeb,R., (1983), "*Strategies of Cooperation in Distributed Problem Solving*", Proc. IJCAI, pp 767-770.

[11] Leveson,N.G., (1990), "*The Challenge of Building Process Control Software*", IEEE Software, pp 55-62.

[12] Hall,L.E., Avouris,N.M. & Cross,A.D., (1990), "*Interface Design Issues for Cooperating Expert Systems*" Proc. 10th Int. Conf. in Expert Systems, pp 455-469, Avignon.

[13] Steiner,D.D., Mahling,D.E. & Haugeneder,H., (1990), "*Human Computer Cooperative Work*", Proc. 10th Workshop on Distributed Artificial Intelligence, Texas.

[14] Roda,C., Jennings,N.R. & Mamdani,E.H., (1990), "*ARCHON: A Cooperation Framework for Industrial Process Control*", in Cooperating Knowledge Based Systems, (Ed S.M.Deen), pp 95-112, Springer Verlag.

[15] Afsarmanesh,H. & McLeod,D., (1989), "*The 3DIS: An Extensible Object-Oriented Information Management Environment*", ACM Trans. on Information Systems, 7 (4) pp. 339- 377.

[16] Lesser,V.R. & Corkill,D., (1983), "*The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks*", AI Magazine, pp 15-33.

[17] Erman,L.D. & Lesser,V.R., (1975) "*A Multi-Level Organisation for Problem Solving Using Many Diverse Cooperating Sources of Knowledge*", Proc. of IJCAI, pp 483-490.

[18] Smith,D. & Broadwell,M., (1988), "*The Pilots Associate: An Overview*", in SAE Aerotech Conference, Los Angeles, CA.

[19] Gasser,L., Braganza,C. & Herman,N. (1989), "*MACE: A Flexible Testbed for Distributed AI Research*", in Distributed Artificial Intelligence (Ed M.N.Huhns), pp 119-153.

[20] Hayes-Roth,F.A., Erman,L.D., Fouse,S., Lark,J.S. & Davidson,J., (1988), "*ABE: A Cooperative Operating System and Development Environment*", AI Tools and Techniques, (Ed M.Richer), ABLEX.

[21] Avouris,N.M., Liedekekerke,M.H.V. & Sommaruga,L., (1989), "*Evaluating the CooperA Experiment*", Proc. of 9th Workshop on Distributed Artificial Intelligence, Seattle.

[22] Agha,G. (1986), "*A Model of Concurrent Computation in Distributed Systems*", MIT Press.

[23] Loingtier,J.M., (1991), "*ALAN: An Agent Language for Cooperation*", IJCAI Workshop on Objects and AI, Sidney, Australia.

[24] Kiss,G., (1991), "*Variable Coupling of Agents to their Environment: Combining Situated and Symbolic Automata*", Proc MAAMAW, Kaiserslautern, Germany.

[25] Agre,P.E. & Chapman,D., (1987), "*Pengi: An Implementation of a Theory of Activity*", Proc. 6th National Conference on AI., pp 268-272.

[26] Brooks,R.A., (1986), "*A Layered Control System for a Mobile Robot*", IEE Journal of Robotics and Automation, 2, 1.

[27] Suchman,L., (1987), "*Plans and Situated Actions: The Problem of Human-Machine Communication*", Cambridge University Press.

[28] Smith,R.G. & Davis,R., (1981), "*Frameworks for Cooperation in Distributed Problem Solving*", IEEE Trans. on System Man & Cybernetics, 11 (1), pp 61-70.

[29] Wittig, T. (Ed), (1992), "*ARCHON: An Architecture for Multi-Agent Systems*", Ellis Horwood, Chichester.