CONNECTIONIST-SYMBOLIC INTEGRATION

From Unified to Hybrid Approaches

EDITED BY

Ron Sun

Frederic Alexandre

Lawrence Erlbaum Associates, Inc., Publishers
10 Industrial Avenue
Mahwah, New Jersey 07430

Library of Congress Cataloging-in-Publication Data

Connectionist-symbolic integration : from unified to hybrid approaches
/ edited by Ron Sun, Frederic Alexandre.
p. cm.
Includes bibliographical references and index.
ISBN 0-8058-2348-4 (cloth : alk. paper). -- ISBN 0-8058-2349-2
(pbk. : alk. paper)
1. Hybrid computers. 2. Systems engineering. 3. Neural networks
(Computer science) I. Sun, Ron, 1960- . II. Alexandre,
Frederic.
QA76.38.C66 1997
006.3--dc21 97-30217
CIP

Books published by Lawrence Erlbaum Associates are printed
on acid-free paper, and their bindings are chosen
for strength and durability.

10 9 8 7 6 5 4 3 2 1

CONTENTS

10

A DISTRIBUTED PLATFORM FOR SYMBOLIC–CONNECTIONIST INTEROPERATION

José C. González
Juan R. Velasco
Carlos A. Iglesias*

Universidad Politécnica de Madrid
** Universidad de Valladolid*
SPAIN

1 INTRODUCTION

Symbolic and connectionist approaches are obviously complementary for problem solving, leading to a growing interest in hybrid systems: those involving the co-operation of both approaches in a single integrated system. However, some general problems may be observed in current literature on the subject. These problems, usual in new research arenas, may render the achievement of advances in the field difficult:

- First, most of the research effort until now has addressed specific applications, generating specific hybrid solutions. Such results rarely evolve toward general models of hybridization.
- Second, applications are usually developed in a purely *ad hoc* manner. Portability, re-usability, scalability and, finally, applicability to real world problems are seriously hampered.
- In this situation, comparing the plausibility, generality and performance of different approaches, models or implementations (in general, any kind of experimental research) becomes an impossible task.

The underlying cause behind this is the lack of theories (conceptual models of reference), methodologies and tools for research and development purposes. Fortunately, some recent works represent considerable contributions in these areas (e.g., Medsker 1995 or, in this volume, chapters 2, 3, and 5).

To address these aforementioned problems, a framework devised for the interoperation of heterogeneous systems is proposed as a well-suited approach. The main characteristics of this approach are identified in section 2. These objectives have nurtured the design of a platform (section 3) and the adaptation of a methodology (section 4) for the development of heterogeneous systems in general, and of symbolic-connectionist hybrids in particular. Some examples are then explored (section 5) and the current research conducted around the platform is overviewed (section 6). A brief summary concludes this chapter.

2 A SOFTWARE INTEROPERATION APPROACH

Symbolic connectionist hybridization represents only one particular case of heterogeneous systems interoperation. A framework suitable for developing the hybrid systems under investigation should at least feature the following capabilities:

- Modularity: Hybrid systems/models should be developed from basic building blocks comprising symbolic, connectionist or fusion modules (those involving the tight coupling of both paradigms), as well as other hardware/software systems (e.g., data acquisition systems, statistical modules, mechanical actuators, etc.)
- Encapsulation: These components should be encapsulated in order to offer homogeneous interfaces.
- Cooperation: Mechanisms suitable for intelligent cooperation should be supported to allow complex interactions among the components of the framework.
- Distribution: Components should be capable of working in distant environments. This requirement can be due to external constraints (such as the execution of individual components in specialized hardware/software platforms), or to the very nature of the problem.
- Ease of use: Any solution to the problem of software interoperation involves some cost, imposing discipline on researchers/developers to force the use of languages, software development methodologies, or tools. However, the overload imposed on researchers to integrate particular pre-existing components in this framework and to organize the interaction of these components should be reduced to a minimum. In any case, such overload should be fully justified in terms of the inherent benefits of this approach.

- Openness: Facilities have to be foreseen for the interoperation of these components with other distributed frameworks currently under development by companies and research institutions.

These considerations lead the authors to propose a multiagent architecture as the most adequate technology for building a common platform for the MIX project (*MIX: Modular Integration of Connectionnist and Symbolic Processing in Knowledge-Based Systems*, European Information Technology Programme, project ESPRIT-9119). An overview of the project objectives and methodology can be found in (Hilario et al. 1994), and a complete description of the MIX platform in (González et al. 1995).

Multiagent architectures belong to the broader field of Distributed Artificial Intelligence. To summarize, we call agents autonomous entities capable of carrying out specific tasks by themselves or through cooperation with other agents. Multiagent systems offer a decentralized model of control, use the mechanisms of message-passing for communication purposes, and are usually implemented from an object-oriented perspective.

3 THE MIX PLATFORM

3.1 Architecture

The MIX architecture consists of two basic entities: the agents and the network through which they interact. The basic functionalities and interfaces of the network constitute the network model. Although different agent models can be integrated, one has been adopted as standard for the current implementation. Moreover, some features are introduced that make this platform especially suitable for symbolic/connectionist hybridization. The starting point for the implementation of the platform has been the formal agent model proposed and implemented by Domínguez (1992).

From an external perspective, an agent is structured as a set of elements:

- Services: functionality offered to other agents.
- Goals: self-imposed tasks (functions that an agent carries out in self-interest, not to meet a demand from another agent).

- Resources: information on external resources (services, libraries, ontologies, groups, etc.)
- Internal objects: data structures shared by all the processes that are launched by the agent to carry out service requests or to achieve goals.
- Control: specification of how service requests are handled by the agent.

At the network level, a yellow-page service is offered by a specialized agent, the *Yellow_Pages* (YP) agent. At birth, agents register with a particular YP agent, giving their net address, and information on the services they offer and the services they might need. Agents can also subscribe to "groups." Groups refer to dynamic sets of agents, and can be used as aliases in service petitions. Thus, these petitions can be addressed to an individual agent, to the agents subscribed to a group, or to all the agents offering the service. The YP agent responds to a registration request providing all the information that an agent needs to know (e.g., the addresses of the agents offering services that it will request). Such information is updated continually by the YP agent.

A YP agent acts as a sort of active repository, not as a router. It registers and diffuses information regarding the structure of a set of agents who cooperate in a particular application. Therefore, different YP agents can be simultaneously active, even in the same machine (provided they use distinct ports for communication). By using the information received from the YP agent, the remaining "application" agents can establish direct communication links, thus making the risk of network collapse (due to saturation of the communication channels in the YP agent) negligible.

Several communication primitives are implemented, including different synchronization mechanisms (synchronous, asynchronous and deferred communications) and higher level protocols, as Contract Net (Smith 1980).

The relationship between the MIX architecture and other models designed for agent interoperation is explored in depth in (González et al. 1995), in particular with AOP: Agent Oriented Programming (Shoham 1993), and with the work around KQML: Knowledge Query and Manipulation Language (Finin and Fritzson 1994). On the other hand, the MIX network model is related to those of EMMA: Enterprise Modeling and Management Architecture (Sycara and Roboam 1992), and the I^3 Reference Architecture (Genesereth and Ketchpel 1994).

3.2 The Knowledge Level

One important problem regarding the exchange of messages in a set of agents is how to facilitate the mutual understanding of the content of these messages. The adopted solution permits the inclusion of a parameter in message headers expressing the language used to codify its content. Moreover, the content can make reference to concepts in an ontology shared by the sender and the recipient agents alike. The current implementation includes tools for the automatic translation of messages written in a reduced version of CKRL (Common Knowledge Representation Language) both to and from C++ code. CKRL was designed by the MLT consortium (project ESPRIT-2154) as an interchange language for symbolic machine learning algorithms (Causse et al. 1993).

On the other hand, it is always possible to compose the content of messages in a free format. But, in this case, sender and recipient need to agree previously on the structure of the body of the messages that they interchange.

3.3 Agents Specification: MIX-ADL

With the MIX project in mind, a specialized language (Agent Description Language, MIX-ADL) has been designed to simplify the specification of the agents cooperating in solving a problem in a hybrid framework.

A MIX-ADL file consists, briefly, of two sections:

- A preamble containing headers, needed predominantly to specify the locations where relevant resources can be found:
  - Net address of the machine where the YP agent for all the agents in the file will be found.
  - Default language that will be used to codify messages sent by all the agents included in the file.
  - Default ontologies for concepts in the application domain.
  - C++ source or object libraries used for low-level service implementation.
- The body of the MIX-ADL specification describes the structure of a set of agents. Agent classes and instances can be declared, leading to a hierarchical organization with inheritance of structure, in the style of the object-oriented paradigm.

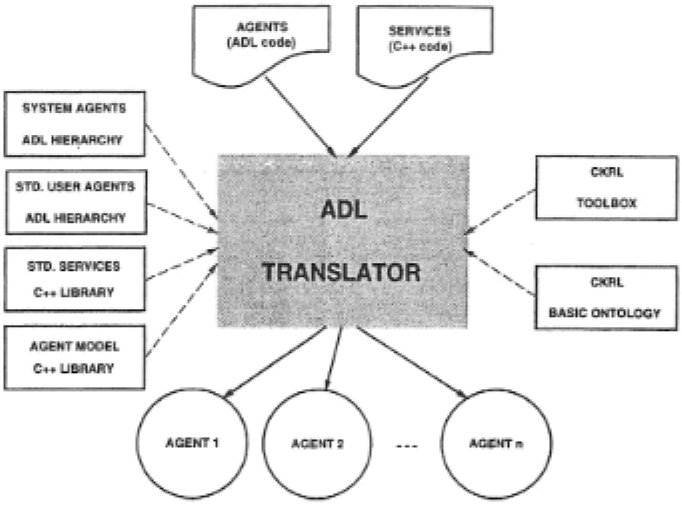

Figure 1 The MIX platform.

3.4 The MIX ToolBox

Finally, some tools have been implemented to translate the MIX-ADL description files to standard C++ programs and for subsequently compiling them. A scheme of the platform is shown in figure 1. It is made up of four elements:

- ADL translator: this translates the MIX-ADL description of a particular set of agents into a set of independent executable programs (one program per agent). These agents can cooperate in the same application, possibly together with other agents compiled in the same or other machines in a distributed and heterogeneous environment. In this task, the translator uses the following three complementary sources.

- MIX C++ libraries: these offer the basic low-level functionality of the platform, including all the typical mechanisms of multiagent libraries as well as general purpose objects and functions.
- CKRL toolbox: this serves for translating CKRL descriptions to C++ classes and objects and C++ objects to CKRL descriptions.
- Standard ADL agent definitions and CKRL ontologies.

Users of the platform have to write the ADL specification of their applications and, eventually, a library containing some objects and functions that implement services, goals, etc. The result obtained from the platform using these elements is a set of executable programs: the agents.

3.5 Types of Coupling

One of the most appealing features of the MIX architecture from the point of view of symbolic–connectionist integration is the possibility of dealing with two levels of integration. By default, our agents are loosely coupled. It means that inter-agent communication is carried out via message passing. However, a more tight coupling mechanism is often needed (mainly for the sake of efficiency). This happens when there is a continuous flow of interaction among agents. Such is the case of the SETHEO system (Goller 1994; Suttner and Ertel 1990).

SETHEO is a theorem prover that uses a neural module to guide the search for a solution across the problem space search tree. When the prover arrives at a situation where several paths could be tried, the neural module suggests the most promising branch to follow.

The MIX-ADL language permits the specification of a group of agents that will be treated as a strongly coupled society. Only part of the services offered by the society as a whole are exported (and thus made visible from the outside). The remaining internal services are offered only to the agents in the society. The petition of these internal services is executed by the platform as a function call instead of using message passing. However, the specific method used for service handling is kept hidden from the user.

4 A METHODOLOGY FOR MULTI-AGENT HYBRID SYSTEMS DEVELOPMENT

As stated in section 1, lack of methodology is a serious drawback in current hybrid systems development. In the MIX project, the CommonKADS methodology (Breuker and Van de Velde 1994; Wielinga et al. 1992) (another product of an European project, ESPRIT–5248) has been adopted and adapted to specific needs in the field of hybrid systems.

CommonKADS is a model-based approach to the development of knowledge-based systems. In this approach, knowledge acquisition is no longer viewed as the simple transfer of knowledge from a human expert to a computer system, but rather as a modeling process. The result of this process is a set of complementary models: Organization, Task, Agent, Communication, Expertise and Design models. The Expertise model plays a pivotal role in the methodology. It represents the problem solving knowledge used by an agent to perform a task.

Obviously, CommonKADS was not originally conceived as a conceptual framework for the development of distributed systems. This is the reason behind the problems found in adopting the methodology for hybrid systems development. The main drawbacks detected concern Agent and Communication models. Firstly, CommonKADS takes into consideration only two kinds of agents: knowledge-based systems and users. But the most difficult problems arise in relation to the Communication model: (1) It only considers human–computer interaction. (2) It does not deal with multi-partner transactions in a natural way. (3) It does not permit a flexible and dynamic assignment of tasks to agents. These are the most important aspects of the original methodology for which modifications have been required. A fully developed example of application of this methodology to the analysis, design and implementation of a hybrid fuzzy-connectionist system can be found in (Iglesias et al. 1996).

5 EXAMPLES

5.1 A Connectionist Agent

In this section, one of the many possible ways of encapsulating a neural net in a MIX agent is shown. This agent is not proposed as a general model, but as an illustrative example. A neural net can be viewed as a reactive agent offering a set

- Fuzzy sets are extracted inductively from the past experience by neural nets and by fuzzy clustering algorithms, both methods being integrated in an intelligent stand-alone hybrid model.
- The complete problem space is partitioned into a set of regions, each one with its own associated fuzzy knowledge base. The identification of regions is carried out by neural nets (Kohonen's self-organizing maps) and fuzzy clustering algorithms
- Fuzzy rules are acquired inductively (in each region of the problem space) by neural nets and by fuzzy techniques, integrated in an intelligent stand-alone hybrid model.
- In a future version, fuzzy sets will be tuned (refined) on line, for a continuous adaptation to the real processes under modeling.

An *intelligent stand-alone model* is a compound of an agent *manager* and a set of learning agents which are joined to form a public group. The *manager* selects the best acquired knowledge base. It can also decide the different regions in which the learning agents should work. Depending on the results obtained by the different learning agents, the *manager* can learn which agents work better under certain circumstances. Thus, the *manager* performs meta-processing. The fuzzy-neural cooperative model consists of three instances of this basic intelligent stand-alone model, and a protocol is defined for communication between the *managers* of the different parameters to be optimized. The main difference with the simple stand-alone model is that first one selects a certain learning agent for working in one region, but the global knowledge base is built from different learning agents. These learning agents can be pure symbolic, pure neural or hybrid (fusion) agents. Each region can be optimized by different agent sets.

6 CURRENT WORK

Release 1.2 of the MIX platform is already available via anonymous ftp from ftp://ftp.gsi.dit.upm.es/pub/mix/. This platform is currently being used by the MIX consortium to test different hybrid models for three different applications. The first one is the optimization of a motor/gear-box combination for a turbo-charged engine. The second one consists of the control of a roll-mill in a steel making company (Pican et al. 1996). The last application pertains to the medical domain: a monitoring system for an intensive care unit.

In addition to this, the platform is being used in three other areas: communication network management (Cáncer et al. 1996), distributed control of fossil-fired power plants (Velasco et al. 1996) and natural language processing.

Further improvements are being included or considered for inclusion in future releases of the platform:

- A KQML-compatible set of performatives could be offered. KQML (Finin et al. 1994) is both a language and a protocol designed for interchanging knowledge. It has been developed as part of the ARPA Knowledge Sharing Effort. A KQML API would be used in order to allow the interoperation of MIX and KQML agents.
- A knowledge agent is being implemented with the CLIPS rule-based shell (Giarratano and Riley 1993) capable of interpreting knowledge-oriented performatives.
- A subset of the KIF (Knowledge Interchange Format) language (Genesereth et al. 1992), also developed in the framework of the Knowledge Sharing Effort, is being considered for inclusion as another native language of the platform.
- Regarding the network model, a federation architecture for YP agents is being designed. The reason for this improvement is to assure system functioning in case of failure of the (only) YP agent responsible for a particular application. In such a situation, alternate YP agents would take charge of managing the application agents affected by the problems.

From the point of view of hybridization, the main issues to be addressed in the near future are:

- The development of a sufficiently large and versatile set of agents apt for symbolic–connectionist integration.
- The design of a set of hybrid models capable of representing a significant proportion of the work carried out in this field and suitable for tackling challenging classes of problems.
- The rigorous evaluation of hybrid models in comparison to pure symbolic and connectionist approaches.

The approach presented here should assist in making the relative benefits and strengths of these models evident, thus guiding future research in this field.

7 SUMMARY

A Distributed Artificial Intelligence approach is proposed as a convenient way for designing an open platform for the integration of connectionist and symbolic systems. In this chapter, our particular perspective on the problems faced in the field of connectionist–symbolic hybridization is introduced. Also a multiagent architecture is put forward as a convenient way of addressing these problems and, following on, the software platform developed for the MIX ESPRIT project is presented. Then, an enhanced version of CommonKADS is proposed as a useful methodology for building hybrid systems.

To conclude: comprehensive methodologies and effective tools have an important role to play in the challenges faced by the field of hybrid systems. This work represents an attempt in this direction, offering a solid basis for further research.

ACKNOWLEDGEMENTS

This research is funded in part by the Commission of the European Communities under the ESPRIT Basic Research Project *MIX: Modular Integration of Connectionist and Symbolic Processing in Knowledge-Based Systems*, ESPRIT–9119. The MIX consortium is made up of the following institutions: Institute National de Recherche en Informatique et en Automatique (INRIA–Lorraine/CRIN–CNRS, France), Centre Universitaire d'Informatique (Université de Genève, Switzerland), Institute d'Informatique et de Mathématiques Appliquées de Grenoble (France), Kratzer Automatisierung (Germany), Fakultät für Informatik (Technische Universität München, Germany) and Dept. Ingeniería de Sistemas Telemáticos (Universidad Politécnica de Madrid, Spain).

This work would have never been carried out without the experience accumulated and the tools developed by Mercedes Garijo and Tomás Domínguez in their Multiagent System Model, that constitutes the basis for the MIX agent model. We are also indebted to Jaime Alvarez and Andrés Escobero (from our group) and Marc Vuilleumier (from Université de Genève, Switzerland) for their contribution to the implementation of the platform.

APPENDIX

ADL DESCRIPTION OF A SIMPLE HYBRID

```
#DOMAIN "ijcai-example"
#YP_SERVER "tcp://madrazo.gsi.dit.upm.es:6050"
#COMM_LANGUAGE ckrl
#ONTOLOGY "example.ckrl"

CLASS Neural_Class -> Basic_Class
    RESOURCES
        REQ_LIBRARIES:  "neural_agent.C"
        REQ_ONTOLOGIES: "neural.ckrl, learning.ckrl"
        REQ_SERVICES:   give_history
        SUBSCRIBE_TO:   Learning_Group
    INTERNAL_OBJECTS
        a_net, l_net -> neural::neural-net
        param ->        neural::learning-parameters
        error ->        learning::error-estimate
    GOALS
        learn: CONCURRENT Learning_Function
    SERVICES
        activate: Activation_Function
                  REQ_MES_STRUCT learning::input-data
                  ANS_MES_STRUCT learning::output-data
             COST Error_Estimation_Function
                  REQ_MES_STRUCT learning::input-data
                  ANS_MES_STRUCT learning::error-estim
        ...
END Neural_Class

AGENT YP_Agent -> YP_Class
END YP_Agent

AGENT Interface -> Basic_Class
    RESOURCES
        REQ_LIBRARIES: "inter_funct.C"
        REQ_SERVICES: start_control;
                      finish_in_order
    GOALS
        interface: CONCURRENT Inter_Function
END Interface

AGENT Selector -> Basic_Class
    RESOURCES
        REQ_LIBRARIES: "selec_funct.C"
        REQ_SERVICES:
            give_sample;
            activate
                CONTRACT_POLICY Eval_Function
                    REQ_MSG_STRUCT example::Cost
    SERVICES
        start_control: CONCURRENT Start_Function
END Selector

AGENT Collector -> Basic_Class
    RESOURCES
        REQ_LIBRARIES: "collec_funct.C"
    INTERNAL_OBJECTS
        history -> History
    GOALS
        get_data: CONCURRENT Get_Data_Function
    SERVICES
        give_sample: Give_Sample_Function
                  ANS_MSG_STRUCT example::Vector
        give_history: Give_History_Function
                  REQ_MSG_STRUCT example::Depth
                  ANS_MSG_STRUCT example::Vector
END Collector

// ******************************************
//                Learning Agents
// ******************************************

AGENT Neural -> Neural_Class
END Neural

AGENT C45 -> Basic_Class
    RESOURCES
        REQ_LIBRARIES: "symbolic_agent.C"
        REQ_SERVICES: give_history
        REQ_ONTOLOGIES: "c45.ckrl","learning.ckrl"
        SUBSCRIBE_TO: Learning_Group
    INTERNAL_OBJECTS
        a_tree, l_tree -> c45::decision-tree
        param -> c45::learning-parameters
        error -> learning::error-estim
    GOALS
        learn: CONCURRENT C45_Function
    SERVICES
        activate: CONCURRENT Decision_Function
                 REQ_MSG_STRUCT learning::input-data
                 ANS_MSG_STRUCT learning::output-data
            COST C45_cost_function
                 REQ_MSG_STRUCT learning::input-data
                 ANS_MSG_STRUCT learning::error-estim
END C45
```

REFERENCES

Breuker, J. and W. Van de Velde (Eds.) (1994). *COMMONKADS Library for Expertise Modelling.* Amsterdam, The Netherlands: IOS Press.

Cáncer, A., J. J. Sánchez, and M. Garijo (1996, April). A multi-agent system for cooperative network-faults management. In *Proceedings of the First International Conference on the Practical Applications of Intelligent Agents and Multi-Agent Technology, PAAM-96*, London, UK, pp. 279–294.

Causse, K. et al. (1993, May). Final discussion of the Common Knowledge Representation Language (CKRL). Deliverable D2.3, MLT Consortium, ESPRIT project 2154, Laboratoire de Recherche en Informatique, Université de Paris Sud, Orsay, France.

Domínguez, T. (1992). *Definición de un Modelo Concurrente Orientado a Objetos para Sistemas Multiagente.* Ph. D. thesis, Dep. Ingeniería de Sistemas Telemáticos, E.T.S.I. Telecomunicación, Universidad Politécnica de Madrid.

Finin, T. and R. Fritzson (1994, July). KQML: A language and protocol for knowledge and information exchange. In *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence*, Lake Quinalt, WA, pp. 126–136.

Finin, T., J. Weber, G. Wiederhold, M. R. Genesereth, R. Fritzson, D. McKay, J. McGuire, R. Pelavin, S. Shapiro, and C. Beck (1994, February). Specification of the KQML agent–communication language plus example agent policies and architectures. Draft document, The DARPA Knowledge Sharing Initiative. External Interfaces Working Group, Baltimore, MD.

Genesereth, M. R., R. E. Fikes, et al. (1992). Knowledge Interchange Format, Version 3.0. Reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, Stanford, CA.

Genesereth, M. R. and S. P. Ketchpel (1994, July). Software agents. *Communications of the ACM, Special Issue on Intelligent Agents 37*(7), 48–53.

Giarratano and Riley (1993). *Clips Manuals, Version 6.0.* Houston, TX: Software Technology Branch, Lyndon B. Johnson Space Center, NASA.

Goller, C. (1994). A connectionist control component for the theorem prover SETHEO. In *Proceedings of the ECAI-94 Workshop on Combining Symbolic and Connectionist Processing*, Amsterdam, The Netherlands, pp. 88–93.

González, J. C., J. R. Velasco, C. A. Iglesias, J. Alvarez, and A. Escobero (1995, November). A multiagent architecture for symbolic-connectionist integration. Technical Report MIX/WP1/UPM/3.2, Dep. Ingeniería de Sistemas Telemáticos, E.T.S.I. Telecomunicación, Universidad Politécnica de Madrid.

Hilario, M., C. Pellegrini, and F. Alexandre (1994, May). Modular integration of connectionist and symbolic processing in knowledge-based systems. In *Proceedings of the International Symposium on Integrating Knowledge and Neural Heuristics*, Pensacola, Florida, pp. 123–132.

Hilario, M. (1997). An overview of strategies for neurosymbolic integration. (Chapter 2 of this volume).

Iglesias, C. A., J. C. González, J. R. Velasco, and K. Eder (1996, January). Prototype of a fuzzy-neural hybrid model. Implementation report. Technical Report MIX/WP2/UPM/2.0, Dep. Ingeniería de Sistemas Telemáticos, E.T.S.I. Telecomunicación, Universidad Politécnica de Madrid.

Khosla, R. and T. Dillon (1997). Task structure and computational level: Architectural issues in symbolic-connectionist integration. (Chapter 3 of this volume).

Magdalena, L. (1997). A first approach to a taxonomy of fuzzy-neural systems. (Chapter 5 of this volume).

Medsker, L. R. (1995). *Hybrid Intelligent Systems*. Boston, MA: Kluwer Academic Publishers.

Pican, N., F. Alexandre, and P. Bresson (1996, February). Artificial neural networks for the presetting of a steel temper mill. *IEEE Expert 11*(1), 22–27.

Shoham, Y. (1993, March). Agent-oriented programming. *Artificial Intelligence 60*(1), 51–92.

Smith, R. G. (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers C-29*, 1104–1113.

Suttner, C. and W. Ertel (1990, January). Using connectionist networks for guiding the search of a theorem prover. Technical Report 342/8/90 A, Institut für Informatik, Technische Universität München, München, Germany.

Sycara, K. and M. Roboam (1992). EMMA: An architecture for enterprise modeling and integration. In N. M. Avouris and L. Gasser (Eds.), *Distributed Artificial Intelligence: Theory and Praxis*, pp. 197–213. Boston, MA: Kluwer Academic Publishers.

Velasco, J. R., J. C. González, C. A. Iglesias, and L. Magdalena (1996, June). Multiagent based control systems: A hybrid approach to distributed process control. *Control Engineering Practice 4*(6), 839–845. (Extended version of a paper previously published in the *Preprints of the 13th IFAC Workshop on Distributed Computer Control Systems, DCCS-95*, A. E. K. Sahraoui and J. A. de la Puente, editors, Toulouse, France, 1995).

Wielinga, B. J., A. T. Schreiber, and J. A. Breuker (1992). KADS: A modeling approach to knowledge engineering. *Knowledge Acquisition 4*(1), 5–53. (Special Issue, The KADS Approach to Knowledge Engineering).