Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Embedded Component Technology for Complex Control Systems [2]

Ricardo Sanz, [1] Carlos Martínez, Manuel Rodríguez and Adolfo Hernando

Autonomous Systems Laboratory
Universidad Politécnica de Madrid
José Gutierrez Abascal 2, 28006 Madrid, Spain

Abstract:
The question of finding the best technology for software-based controller deployment is still open. One of the main problems is that computing infrastructures for controllers are composed by heterogeneous hardware and software platforms scattered over a set of heterogeneous networkig infrastructures. The standardised object-oriented efforts done around the the OMG specifications try to overcome some of the difficulties raised by this heterogeneity. Inside the Distributed Object Computing (DOC) landscape, CORBA is a well known framework for the construction of modularised, object oriented, distributed applications. The CORBA object model, however, is not enough when confronting the problems related to deplyment, configuration and evolutionary maintenance of systems. This paper describes the ECF component technology specifically built for the construction of component-based, distributed embedded systems.

Keywords: Components, Real-Time, Embedded, CORBA, CCM, Distributed Object Computing

1. INTRODUCTION

Control systems are bassically built using software technology. Specialized in the past, in recent years the domain of embedded software is turing into the use of tools and assets that come from the mainstream software engineering domain.

This movement is motivated by the continuous increase in complexity (Sanz et al., 1998) and the changes happened in the embedded systems domain: but the question of finding the best technology for software-based controller deployment is still open. One of the main problems is that computing infrastructures for controllers are composed by heterogeneous hardware and software platforms scattered over a set of heterogeneous networkig infrastructures.

The standardised object-oriented efforts done around the the OMG specifications try to overcome some of the difficulties raised by this heterogeneity. Inside the Distributed Object Computing (DOC) landscape, CORBA is a well known framework for the construction of modularised, object oriented, distributed applications. The CORBA object model, however, is not enough when confronting the problems related to deplyment, configuration and evolutionary maintencae of systems. This paper describes the ECF component technology oriented towards the construction of component-based embedded systems.

In the last years, the embedded systems domain is trying to incorporate a component-based approach to systematise software development. This is mostly notable given that the traditional engineering approach to developing embedded applications has been driven by the primary goal of maximizing performance whilst absorbing the minimum of resources like memory or power consumption. This conventional approach is usually adopted by compromising software engineering productivity desiderata, such as reusability or maintainability, sacrificing modularization to achieve reduced size and/or complexity.

However, even in the most constrained class of embedded applications —as those designed for deployment on DSP devices— software processes are supported by toolchains based on component models that are usually tailored to such devices.

This paper describes the technological products of the IST COMPARE project funded by the European Commission IST Programme. The main objective of the COMPARE project was the provision of models and technology to fulfill real-time requirements in component-based, real-time embedded applications (RT/E). The approach of COMPARE that makes a bigger difference from other real-time component models like Pecos, ACCORD, AOCS, OBS (e.g. et al. (2002)) is the focus on standardisation of the technology following an open process.

2. EMBEDDED COMPONENTS LANDSCAPE

One seemingly unavoidalbe trend in software for real-time embedded systems is towards complexity. This raising complexity is due to the increased functionality these systems are required to to provide in an increased uncertainty environment thay may include other interacting systems.

[1] Corresponding author, ricardo.sanz@upm.es
[2] This work was funded by the European Commission through grant IST-004669 COMPARE.

Real-time software architectures commonly used in embedded systems were primarily meant to fulfil just real-time performance issues and are not suitably structured to support this evolutionary increase in complexity. One big challenge in complex embedded systems is thus to reframe the design process from a performance-centric approach to a complexity-centric approach, whithout loosing the temporal performance so critical for these systems.

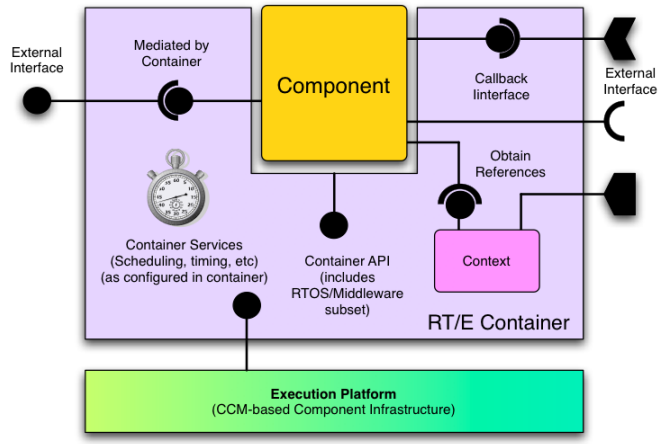

Fig. 1. The ECF Component-Container model for embedded real-time systems.

This movement toward increased complexity is not exclusive of the embedded systems domain but is also happening in conventional software. Non real-time applications are also facing this increase in complexity and have produced some relevant paradigms of interest in the embedded community.

The paradigm we are exploring in the work described in this paper is that of the very promising component/container model. However current models (*e.g.* Sun's EJB [3] or OMG's CCM [4]) were originated and targetted to enterprise information systems. Therefore, the services they specify (*e.g.* persistence or transactions) are not mainstream necessities of real-time and embedded systems engineering.

Perhaps motivated by losing the enterprise role it had due to HTML-based SOA [5] CORBA-related efforts have moved towards real-time and embedded areas (*cf.* most of the recent adopted specifications at www.omg.org).

The recently adopted Lightweight CCM, CCM-based component technology has started a move in this very same direction by defining a minimum profile upon which it is now possible to build a model dedicated to real-time and embedded.

This paper The purpose of this project is to study a framework based on this model (CCM for real-time and embedded), to realise a reference implementation of it, to test it on two different application cases (one based on RT-CORBA, the second on OSEK-VDX RTOS) and to push the results at OMG for standardisation.

[3] Enterprise Java Beans.
[4] CORBA Component Model.
[5] Service Oriented Architecture.

3. PROJECT RESULTS

The COMPARE project ended in December 2006 after two years and a half and the results obtained can be summarized in the following five items:

- The definition of a RT/E framework for component-based embedded systems: this framework is called ECF - *Embedded Component Framework*.
- A reference implementation of the ECF based on RT-CORBA [1].
- A first technology demonstration in the field of Software Defined Radio [5] built with the CORBA [7] reference implementation.
- A derived partial implementation on OSEK-VDX [2].
- A second demonstration on the field of Electricity Management Systems built with the OSEK-VDX partial implementation.

The global objective of the separation of concerns approach in component-based systems is the segregation of business logic from underlying RT platforms to maximize component reusability. This reusability was demonstrated by the mapping of the general ECF model into two quite different deployment platforms: RT-CORBA and OSEK-VDX.

OSEK-VDX and RT-CORBA differ enormously in the level of services they provide and hence it is therefore unrealistic to plan to support the full range of functionalities as the reference implementation does. The intention is to port to OSEK-VDX all that can be ported at a reasonable cost. Obviously it will comprise, as a minimum, what is needed to support the related electricity management use-case.

4. APPROACH

The main technological goals of this work are three-fold:

- full separation of concerns between system functions and enabling software platform technologies (middleware and RTOS), so as to maximize deployability of reusable components across an heterogeneous platform base,
- enabling the use of a variety of architectural styles for embedded systems (including distributed, real-time and deeply embedded devices)
- facilitate tool-supported construction by interface-level composition of reusable modules following standard component models (see Figure **??**).

These directions were identified as necessary by a domain analysis of current common concerns in the domain of real-time and embedded systems design and implementation [3].

The separation of concerns between system functions and enabling software technologies aspect is recognized as an important technological lever to treat portability because in this approach, the system functions implementation are underlying technology free.

Separation of concerns is also natural in the scope of component orientation, which is all about placing the component as the unit of architectural decomposition: traditional approaches for real-time systems focusing first

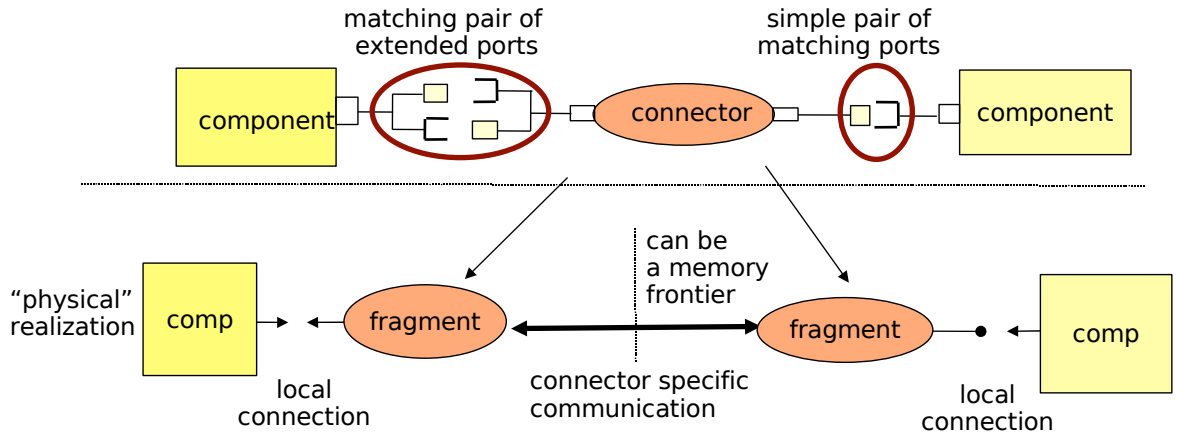

Fig. 2. The ECF is based on a Component/Connector model that enables the definition of the port type needed for specific interaction models and the connector required for interaction between entities having some extended ports and attributes.

on tasks and synchronisation characteristics often resulted in systems shaped by these aspects, implying important coupling between systems functions, complex mapping of functions to tasks, and intricate tasks collaboration.

Supporting a variety of architectural styles is also important in the real-time embedded domain, in which fundamentally different execution models and architectures (most of the time engineering domain oriented) are involved. The motivation for putting emphasis on a clean integration of custom interaction patterns is to avoid relegating these aspects to component programmer themselves in case the component model would have been found limitative. Another key motivation in this regard stems from the fact that intermediate-granularity components (i.e. not too coarse) can reveal the candidates for reuse, and often exhibit between them complex interactions.

Tool-supported composition is also important, as composing and configuring complex systems by hand reveals very difficult and error prone. The basis for the automated composition is an architecture description in the form of a component assembly descriptor. Automated composition is at the heart of component-oriented approaches. It can also be thought as a particular type of separation of concerns, also reducing the complexity to setup and configure a system.

The technical approach, in order to achieve the goals explained above, is based on extending and profiling the Lightweight CCM (LwCCM) specification [4] and COMPARE contributions come in three forms:

- Integration of new features in the CCM component model and deployment facilities.
- Definition of profiles from Lightweight CCM and OMG deployment and configuration.
- Adaptation of the LwCCM specification: some minor needs for modification in the specification.

5. CORE GUIDELINES OF THE ECF APPROACH

Component-based applications are built by component interconnection. Components provide services to other components and require services from other components through provided (facets) and required ports (receptacles) (see Figure **??**). To perform their function, components may use services from the underlying platform (e.g. computing or timing events).

In order to maximise reusability of component business logic the ECF model isolates the component from the execution platform by means of a container that provides a set of technical services that are used by the component. The ECF implementation is open and extensible and these technical services are provided by means of container-level pluggable elements [6].

Many services are under development in the context of the COMPARE project in order to provide some basic functionality required by the demonstrators. These services include for example distributed state machines, timers, clock synchronisation, transactional memory, distributed logging, etc.

With this approach the ECF can separate the functional aspects provided by the component from the non-functional aspects concerning its use in a particular application that are provided by the container (see Figure 2).

Detailed specifications about the framework including the proposed extensions to LwCCM specifications are provided in the framework design documentation from the COMPARE project [10].

6. DEVELOPMENT PROCESS

The development process proposed by COMPARE and outlined outlined in Figure **??** is obviously focused on component-based reuse, separating the engineering activity in three major phases:

(1) Production of components
(2) Application design
(3) Component deployment

The developed approach not only focuses on an optimised and fully modular realisation of the needed software features, but also to a development process enjoying the following properties:

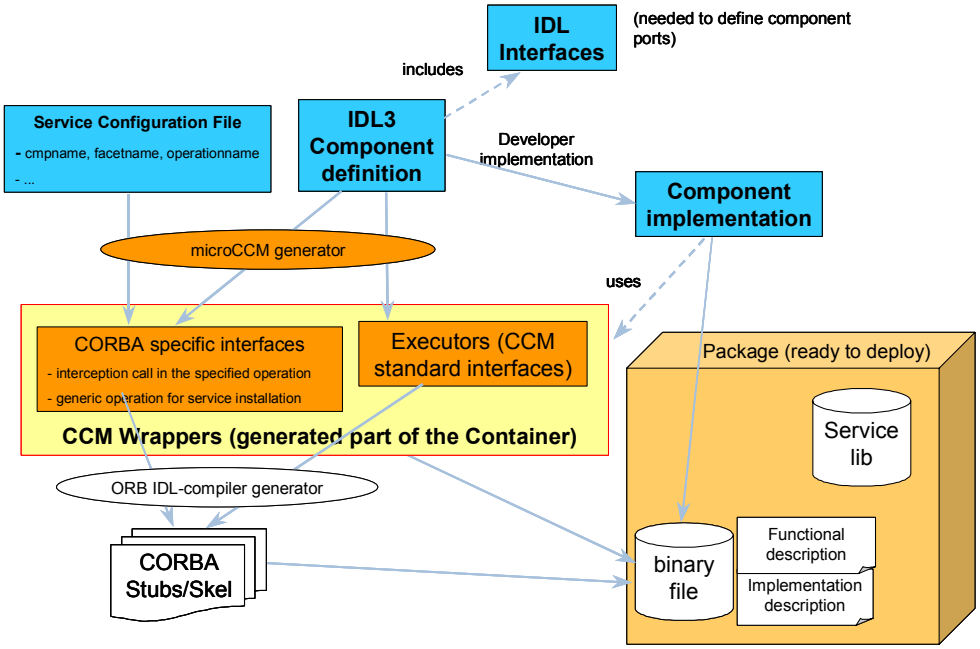

Fig. 3. The ECF engineering process for embedded real-time systems.

- traceability of the software architecture (description of the breakdown of the application into components) all along the development process,
- separation of concerns in the implementation phase: realising the decided execution semantic on an RTOS or middleware platform, versus implementing and capitalising on domain functional blocks integratable in fundamentally different target platforms in a guaranteed a priori manner,
- ease and agility in the final integration phase of the system, by allowing to relocate lately components from one device to another.

A critical aspect in this engineering process is component selection (see Figure **??**).

The current implementation of the Embedded Component Framework includes software for infrastructure and containers — the latter comprising building support (toolchain) and the provision of various component administration functions.

The testbed employed in the development of the ECF and the evaluation of its real-time properties is comprised of a General Purpose Processor (GPP) and a Digital Signal Processor (DSP) device. They represent the hardware used as part of the signal processing chain of a Software Defined Radio (SDR), one of the demonstrators of the project.

The application software is built following a functional decomposition of an SDR waveform, represented as generated components, and distributed across the GPP and DSP devices using the ECF framework.

The framework hence shows the following capabilities of the component architecture,

- Support for the composition of an application by the assembly and deployment of components onto resource-constrained devices.
- Realize a real-time capability by use of the COMPARE component execution environment and the underlying real-time Operating System (RTOS) and appropriate middleware infrastructure for distributed communications.
- Demonstrate a level of adaptability such that components can be developed for diverse platform RTOS, communications data transport, and programming languages.

7. ADVANTAGES OF THE ECF TECHNOLOGY

The advantages of using component-based technology in the RT/E domain, are founded in the core architecture of the component model, and are realized using various development and productivity tools specifically adapted to this class of application.

Portability : RT/E applications are developed for a wide diversity of operating systems (OS) and hardware platforms. One commonly adopted approach to providing more portable solutions is to incorporate an application layer that abstracts the required underlying platform functionality. Unfortunately, this approach presents particular difficulties when defining an API having common semantics across widely diverse architectures, and it may be the case that useful (often proprietary) functions needed to support a particular business goal may be unavailable on some supported platforms. The complexity of the abstraction layer will also increase with the diversity of platform support required, therefore the solution will not scale well when introducing new OS and hardware variants. The core abstractions in the component model on the other hand, provides the opportunity

to develop specialized (fit-for-purpose) containers, and allows the re-deployment of components from one RT/E device to another without the need for source-code modification.

Reusability : The reuse of software units within the scope of an organisation or collaborating parties is a practice known to increase cost-effectiveness in software development (see Figure **??**). However, in order to exploit such advantages, an appropriate level of granularity is necessary for the decomposition of application code into re-usable units needs to be identified. In an Object Oriented programming model, difficulties can arise due to the assumption that different entities in a software system/archive have interfaces that are amenable to inheritance and aggregation — also that these are written using the same programming language. Thus, connecting large numbers of relatively homogeneous units using specialised coding conventions and styles can be error-prone and expensive. In a component model, formal definitions strengthen the visibility of component dependencies and therefore enforce rules for application assembly. This has the overall effect of minimizing ambiguity when selecting software units for re-use, thereby increasing cost-effectiveness in development, and reducing time-to-market.

Separation of Concerns : In conventional software development, there is often an inherent need to understand the broader issues associated with non-functional systems - enabling technologies, software integration, global system architecture etc. Domain experts therefore, can rarely invest their energy exclusively into the design development of business solutions. This problem is also exacerbated where the complexity of non-functional systems increases — as is the case with RT/E applications. Here, detailed (specialist) knowledge of non-functional systems is tightly coupled with application development concerns. The separation of concerns evident in the COMPARE approach provides the opportunity for domain experts to focus exclusively on the design development of business solutions, resulting in a more cost-effective use of development resources.

Visibility : Software development practices that re-use in-house coding styles — naming conventions, layout, modularization, and language specific productivity techniques, make it difficult for external parties to engage in collaborative development projects. Where new products are developed, this situation would likely impact to a greater extent on time-to-market. In a standards-based component model, coding styles/patterns are formally specified - usually public domain. This arrangement fosters cleaner integration and encourages closer partnerships between collaborating parties.

Quality in Development : When the use of conventional programming leads to the creation of monolithic systems, additional complexity is introduced into testing and validation systems. Consequently, these are relatively expensive to produce and are often not re-usable. The de-composition in component-model architectures, allows software units (components) to be naturally isolated for testing and validation. This provides the opportunity to adopt an evolutionary approach to test development that can be integrated throughout the software development life cycle.

Quality in Production: Strategies for self-management that ensure high system availability can be applied more systematically using components than with other more conventional systems. A component could for example, validate that an input value is within an acceptable range and provide a response that is non-intrusive to the component. Similar self-management strategies can improve system reliability by safeguarding component integrity in production.

8. A NOT SO SIMPLE EXAMPLE

Martinez [11] demonstrates the advantages of the ECF approach in the implementation of a component-based simple —demonstration-class— adaptive controller.

The extended component technology provided by the COMPARE ECF makes possible the minimally intrusive interception of component calls to exploit the access to call/return values for the implementation of added functionality to preexisting systems.

Using this approach only latency and jitter are introduced; there maximizing reusability as there is no need of component code instrumentation. This makes possible the use of this container-level interception technology for externally sourced, binary distributed components (or even for the reuse on non-compoanes-based controller implementations).

This transparent access to interaction flows can be used for component introspection or reflection, hence increasing the flexibility and reusability of of the codebase by means of standardised design patterns.

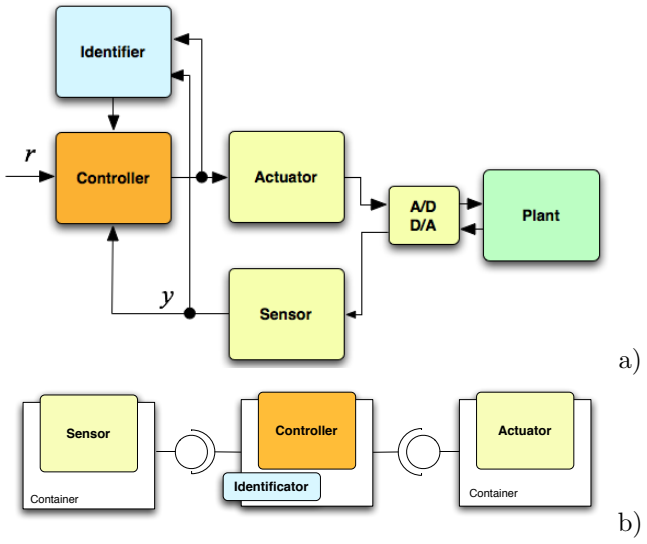

Fig. 4. A block diagram (a) of a simple adaptive controller to be built using three ECF components (*controller, sensor and actuator*) and a container-pluggable service (*the identificator*). In (b) we can see a component diagram of the three ECF components and the container-pluggable service (*the identificator*) that uses container level interception to perform non-intrusive control system monitorisation.

The simple adaptive controller of this example [11] uses three full-fledged real-time ECF components: *controller,*

sensor and actuator and employs a pluggable service in the controller container —the *identificator*– (see Figure 5) to implement an adaptive controller. This controller uses container level interception —in the controller container because this is the component that fixates the control period— to perform non-intrusive control monitorisation.

This same structure —a design pattern— can be used to implement a plant-adptive controller (e.g. a classical model-reference adaptive controller) or an infrastructure-adaptive one (e.g. to exploit processor feedback scheduling).

9. CONCLUSIONS

By the end of the project the COMPARE Embedded Component Framework and the two implementations (the RT-CORBA reference implementation and the OSEK-VDX reduced implementation) were available for test in the embedded systems community. Two demonstrations based on both implementations were built in the fields of Software Defined Radio and Electricity Management Systems.

For each demonstration, the work was separated in i) integration of the base software platform on the selected hardware (to properly interface application components to the hardware capabilities of the target) and ii) architecture and realization of the applications themselves.

In the software defined radio, the ECF technology has been experimented from voice and data input-output down to the radio transceiver (interface with the analog RF part of the radio) to implement a public test waveform having hard real-time constraints. It has been implemented using the component framework implementation based on RT-CORBA.

The second use-case has consisted in the realisation of a software electrical breaker system prototype, incorporating power-line metering, signal anomaly detection, and smart-I/O for reaction. It has been implemented using the component framework implementation based on OSEK / OSEK-Com.

Two additional proofs-of-concept in the field of distributed process control systems and mobile robotics continue under development in our own laboratory.

As of the time of this writing there exists an alpha version of the framework and some of the services that can be requested for experimentation from the COMPARE Consortium Dissemination Team. More information can be found in the project website `www.ist-compare.org`.

REFERENCES

Paul Czerny. Current uses of middleware in embedded/real-time applications. In *1st OMG Workshop on Real-time and Embedded Distributed Object Computing*, Falls Church, USA, 2000.

M. Winter et al. Components for embedded software — the pecos approach. In *Second International Workshop on Composition Languages*, Málaga, Spain, June 2002.

Hermann Kopetz. *Real-Time Systems – Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, 1997.

Douglas Locke. Real-time linux – what is it, why do you want it, how do you do it?, 2000. http://www.linuxdevices.com/articles/AT6090565653.html.

OMG. Common object request broker architecture and specification. release 3.0.3. Technical Report Document - formal/04-03-12, Object Management Group, Falls Church, USA, 2004a.

OMG. Corba/e final adopted specification. Technical Report Document – ptc/06-08-03, Object Management Group, Falls Church, USA, 2004b.

OMG. Real-time CORBA specification. release 1.2. Technical Report Document - formal/05-01-04, Object Management Group, Falls Church, USA, 2005.

OMG. Corba component model. release 4.0. Technical Report Document - formal/06-04-01, Object Management Group, Falls Church, USA, 2006.

Ricardo Sanz. Agents for complex control systems. In Tariq Samad and John Weyrauch, editors, *Automation, Control, and Complexity: New Developments and Directions*, chapter 10, pages 171–190. John Wiley and Sons, Chichester, UK, 2000. ISBN 0-471-81654-X.

Ricardo Sanz. Mapping IEC 61850 to CORBA. In *Proceedings of the OMG Workshop on Embedded & Real-Time Distributed Object Systems*, Burlingame (CA), USA, January 7 - 10 2002.

Ricardo Sanz and Mariano Alonso. CORBA for control systems. *Annual Reviews in Control*, 25:169–181, 2001.

Ricardo Sanz, Miguel J. Segarra, Angel de Antonio, José A. Clavijo, and Idoia Alarcón. Issues in the development of software intensive control systems. In *Proceedings of COSY Anual Workshop*, Ohrid, Macedonia, October 1998.

Ricardo Sanz, Idoia Alarcón, Miguel J. Segarra, Angel de Antonio, and José A. Clavijo. Progressive domain focalization in intelligent control systems. *Control Engineering Practice*, 7(5):665–671, May 1999a.

Ricardo Sanz, Fernando Matía, and Eugenio A. Puente. The ICa approach to intelligent autonomous systems. In Spyros Tzafestas, editor, *Advances in Autonomous Intelligent Systems*, Microprocessor-Based and Intelligent Systems Engineering, chapter 4, pages 71–92. Kluwer Academic Publishers, Dordretch, NL, 1999b. ISBN 0-7923-5580-6.

Ricardo Sanz, Miguel J. Segarra, Angel de Antonio, and José A. Clavijo. ICa: Middleware for intelligent process control. In *IEEE International Symposium on Intelligent Control, ISIC'1999*, Cambridge, USA, 1999c.

Ricardo Sanz, Miguel Segarra, Angel de Antonio, Idoia Alarcón, Fernando Matía, and Agustín Jiménez. Plant-wide risk management using distributed objects. In *IFAC SAFEPROCESS'2000*, Budapest, Hungary, 2000.