A Reconfigurable Concurrent Function Block Model and its Implementation in Real-Time Java

Xu, Y.; Brennan, R.W.; Zhang, X.; Norrie, H.

A Reconfigurable Concurrent Function Block Model and its Implementation in Real-Time Java. *

Xu, Y., Brennan, R.W., Zhang, X. and Norrie, H.
2002

* published in: Journal of Integrated Computer-Aided Engineering, Volume 9 (2002), Pages 263-279 (In press). NRC 44949.

Integrated Computer-Aided Engineering 9 (2002) 263–279
IOS Press

A reconfigurable concurrent function block model and its implementation in real-time Java

Robert W. Brennan[a,*], Xiaokun Zhang[b], Yuefei Xu[c] and Douglas H. Norrie[a]
[a]*Department of Mechanical and Manufacturing Engineering, University of Calgary, Calgary, Alberta, Canada T2N 1N4*
[b]*Centre for Computing and Information Systems, Athabasca University, 1 University Drive, Athabasca, Alberta, Canada T9S 3A3*
[c]*Institute for Information Technology, National Research Council of Canada, 1500 Montreal Road, Bldg. M-50, Ottawa, Ontario, Canada K1A 0R6*

Abstract: This paper focuses on the important holonic manufacturing systems issue of automatic and dynamic adaptability to change at the physical machine level of control. We propose a model to support configuration and reconfiguration of real-time distributed control systems that is built upon recent models for distributed intelligent control and provide an example of its implementation on a real-time Java platform.

1. Introduction

A key goal of holonic manufacturing systems (HMS) research (and arguably, its central ideal) is to develop manufacturing systems that can automatically and dynamically adapt to change. For example, since the HMS Consortium's [19] inception in 1992, its goal has been "to attain in manufacturing the benefits that holonic organisation provides to living organisms and societies, e.g., stability in the face of disturbances, adaptability and flexibility in the face of change, and efficient use of available resources." [36].

Researchers from various communities (e.g., operations research, artificial intelligence, agent system, holonic systems) have been tackling the general question of developing this type of "adaptive" or "agile" manufacturing system for a number of years. Recently, it has become apparent that systems composed of autonomous and cooperative entities (e.g., autonomous and cooperative software agents and holons) show the most promise as an approach to this general problem. For example, in the area of product design, researchers have successfully applied this concept to enable teams of designers to work together over widely distributed locations to design complex products using a variety of design and analysis tools (e.g., PACT [8], ACDS [9], RAPPID [30]). Similarly, in the area of enterprise integration and supply chain management, agent-based and holonic concepts have been applied to manage world-wide networks of suppliers, factories, warehouses and distributors to ensure that the fundamental manufacturing requirements of material acquisition, transformation and delivery are efficiently achieved (e.g., ISCM [16], AIMS [28], CIIMPLEX [31]). As well, the notoriously difficult problem area of manufacturing planning and scheduling has been of particular interest to multi-agent and holonic systems researchers (e.g., AARIA [2], MASCADA [6], MetaMorph [27], YAMS [29], PROSA [37]).

Another important area of research is that of real-time distributed control: i.e., the development of approaches to enable real-time distributed systems to automatically reconfigure to adapt to changes in the

*Corresponding author.

ISSN 1069-2509/02/$8.00 © 2002 – IOS Press. All rights reserved

manufacturing environment. For example, in conventional real-time industrial control systems (e.g., programmable logic controllers (PLC)), reconfiguration involves a process of first editing the control software offline while the system is running, then committing the change to the running control program. When the change is committed, severe disruptions and instability can occur as a result of high coupling between elements of the control software and inconsistent real time synchronisation. For example, a change to an output statement can cause a chain of unanticipated events to occur throughout a PLC ladder logic program as a result of high coupling between various rungs in the program; a change to a proportional-integral-derivative (PID) function block can result in instability when process or control values are not properly synchronised. Clearly, new software approaches are required to avoid these coupling and synchronization problems when real-time distributed control applications are initially configured and later reconfigured in response to change.

As noted previously, both agent-based and holonic approaches can be applied to achieve adaptable systems at the higher, non-real-time or soft real-time planning levels of a manufacturing enterprise. However, to date all of the work on applying these concepts to the real-time control domain has only focused on implementing agents and holons at this level and not on the central holonic issue of adaptability. For example, research has been conducted into holonic models for distributed real-time control (e.g. [14,41]), distributed real-time operating systems (e.g. [3,40]), verification techniques (e.g. [38]), and robotic cell control (e.g. [18]).

This limitation of holonic systems research becomes even more apparent when one considers the key differences between agent and holonic systems concepts. Multi-agent systems is for example, can be thought of as a general software technology; manufacturing however is fundamentally concerned with "ironware" (i.e., physical equipment and products). As a result, when these two worlds merge, it becomes useful to start thinking about a different type of agent than those that populate classic agent-based systems. In particular, it becomes useful to think about "physical agents" or "holons" that possess both classical software agent capabilities as well as the capability to interface with physical equipment on the shop floor. Since the physical equipment that populates a manufacturing facility is typically classified as real-time (i.e., these systems tightly link correctness with timeliness [10]), real-time distributed control (and reconfiguration of these systems) is of particular relevance to holonic systems research.

In this paper we address this important holonic systems research issue and propose a model to support configuration and reconfiguration of real-time distributed control systems that is built upon recent models for distributed intelligent control. We start with a description of these existing models for real-time distributed control in Section 2. Next, we discuss the drawbacks of these models with respect to the holonic goal of adaptability, and propose a general approach for reconfiguration. In Section 4, we describe our approach in detail then provide an example of its implementation in real-time Java in Section 5. Finally, we conclude with a discussion of the benefits of this approach as well as our current work on extending our model.

2. Background

In this section we provide a brief overview of the current models for real-time distributed control. We start with a general overview of the research in this area then focus on the IEC 61499 model for distributed industrial process control [21]. The authors' reconfiguration model that is reported in the remainder of this paper is linked closely to, and can be considered as extending, the IEC 61499 model described in this section.

2.1. Distributed intelligent control

Distributed intelligent control involves matching the control model more closely with the physical system. This is particularly relevant to manufacturing control systems that are required to control widely distributed devices in an environment that is prone to disruptions. With this model, control is achieved by the emergent behaviour of many simple, autonomous and cooperative entities that are based on the principles of object-oriented and agent-based systems.

Although there has been a considerable amount of work on agent-based approaches to the upper/planning and scheduling level of control very little work has been done on applying these techniques to the lower, real-time control level. The main barriers at the real-time control level result from the difficulty of implementing multi-agent systems (MAS) concepts in a stochastic environment where hard real-time constraints must be met to achieve safe system operation.

The primary distinction between non-real-time and real-time systems is that real-time systems tightly link correctness with timeliness. In other words, deadlines must be met under hard real-time (i.e., tasks must finish by a specified time) and soft real-time (i.e., tasks must meet deadlines on average) constraints [11]. As well, real-time systems are typically safety-critical systems (i.e., the system should not incur too much risk to persons or equipment), and as a result, characteristics such as timeliness, responsiveness, predictability, correctness and robustness are of fundamental importance. Because of the more stringent requirements for latency, reliability and availability, it follows that the step from the non-real-time or soft real-time domain is a large one, requiring new models and methodologies for distributed control.

Recently, there have been a number of advances in this area that provide the tools to move away from the traditional centralised, scan-based programmable logic controller (PLC) architecture towards a new architecture for real-time distributed intelligent control. In particular, there have been a number of advances recently in programming languages [23], models for distributed control [21] and software methodologies [26] that are relevant.

Several authors, particularly those associated with the Holonic Manufacturing Systems (HMS) consortium, have carried-out a considerable volume of relevant research. This includes Sieverding [34] and Bussmann and Sieverding [7] who addressed the issues of an agent-based HMS and data test missions for their control systems, although it was Zhou et al. [41] who identified and rigorously defined the application of real-time principles to agent-based HMS.

Many authors have also studied the control of processes in non-holonic manufacturing (e.g. [11,22]). In general, these analyses are based on specific logic formalisations and are applied in particular engineering domains. Moreover, the management of such processes in real-time is used throughout the systems entire specification, design and maintenance. We feel that this metaphor would create a conflict with the emerging practice in HMS, namely that of developing the system as the seamless integration of real-time and non-real-time control components. Hence we need a unified infrastructure to support reconfiguration throughout the various echelons of functionality in an HMS.

2.2. *The IEC 61499 standard*

The International Electro-technical Commission (IEC) 61499 standard addresses the need for modular software that can be used for distributed industrial process control [21,24]. In particular, this standard builds on the function block portion of the IEC 61131-3 standard for PLC languages [23] and extends the function block (FB) language to more adequately meet the requirements of distributed control in a format that is independent of implementation.

IEC, with the help of several HMS consortium members, have developed the Function Block Architecture as a new standard to model industrial process control systems using decentralisation and hard real-time design philosophies. The architecture permits access to the controlled manufacturing process via an IEC 61499 system that contains an organisation of devices. The four main models of the IEC 61499 standard are illustrated in Fig. 1. These models are organised in increasing levels of granularity: system model (Fig. 1(a)), device model (Fig. 1(b)), resource model (Fig. 1(c)), and application model (Fig. 1(c)).

A holon is represented by one or more hardware devices and can interact via one or more communication networks. As shown in Fig. 1(b), each device comprises of one or more resources (i.e., processor with memory) and one or more interfaces. Interfaces enable the device to interact with either the controlled manufacturing process (via a process interface) or with other devices through a communication interface.

Resources are logical entities with independent control over their operations including the scheduling of their tasks. A resource can be created, configured etc. (as part of the system's life-cycle) via a management model.

Applications (software functional units spanning one or more resources and over one or more devices) are networks of function blocks (FB) and variables connected by data and event flows. Such applications aid the modelling of cooperation between the autonomous holons. Function blocks receive events/data from interfaces, process them by executing algorithms and produce outputs, all handled by an event control chart.

Function blocks' algorithms can be written in either high-level programming languages (e.g., C++) or in the IEC 61131 languages for programmable controllers (e.g., Ladder Diagrams, Structured Text). A distribution model controls how applications are decomposed while ensuring that every function block is an atomic unit of distribution.

2.3. *IEC 61499 function blocks*

An example of an IEC 61499 function block is shown in Fig. 2. As was mentioned previously, the function block can be thought of as an "enhanced" object. Objects can be thought of as entities that do things (i.e., we

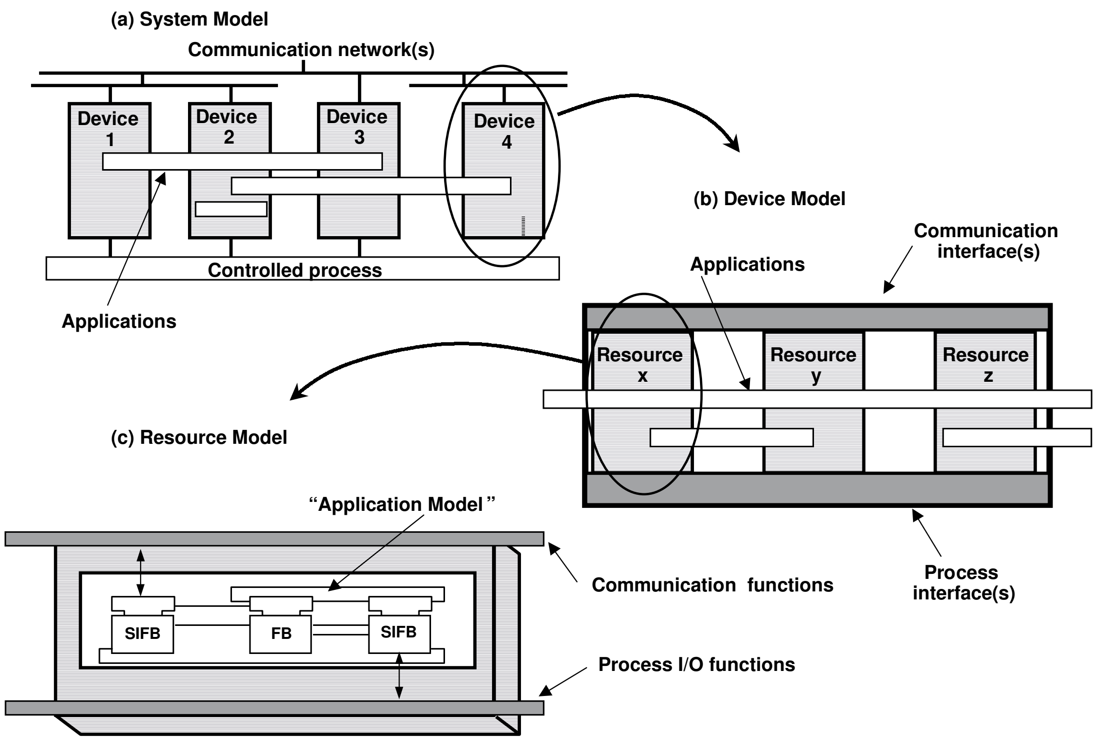


Fig. 1. The IEC 61499 system, device, resource and application models.

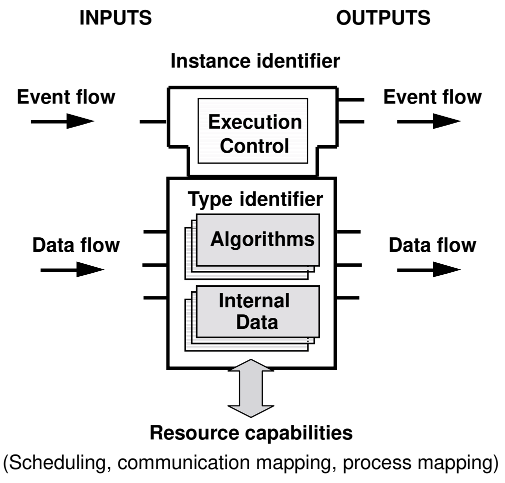


Fig. 2. IEC 61499 function block model.

send a message to them and ask them to perform what they "do"). Function blocks use very specific kinds of messages that are important to the control domain: i.e., events or, in other words, instantaneous occurrences that are used to schedule the execution of the algorithm. As well, data is also passed between function blocks (e.g., messages containing text, floating point numbers, Boolean numbers etc.), which we would expect from a conventional object.

As one would expect, the function block achieves both algorithmic and data abstraction (algorithms and internal data) typical of a conventional object. Looking at the top of the function block though, one can see that events are related to the execution control portion of the function block. This portion of the function block is concerned with scheduling the execution of the function block algorithm's operations on a resource. When the algorithm has completed its execution, execution control generates zero or more event outputs as appropriate.

In order to gain a clearer understanding of the IEC 61499 function block, we can look at the data/event synchronisation model shown in Fig. 3. This figure shows at a typical scenario where an event arrives that initiates the execution of an algorithm. The process is synchronised as follows [21]:

1) Relevant input variable values are made available.
2) The event at the event input occurs.
3) The execution control function notifies the resource scheduling function to schedule an algorithm for execution.
4) Algorithm execution begins.
5) The algorithm completes the establishment of values for the output variables.

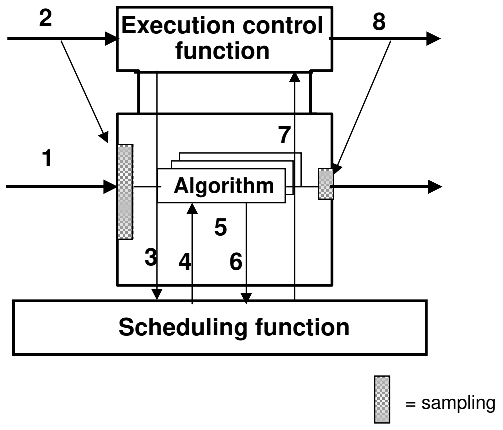


Fig. 3. Data/event synchronization model.

6) The resource scheduling function is notified that algorithm execution has ended.
7) The scheduling function invokes the execution control function.
8) The execution control function signals an event output.

As was noted previously, execution control (EC) is responsible for the synchronisation of the function block. Figure 4 shows an example of an execution control chart (ECC) that is used to specify the sequencing of function block algorithms.

The ECC is included in the execution control block section of the function block and contains one EC initial state and one or more EC states. Each EC state can have zero or more EC actions, which have an associated algorithm and an event to be issued on completion of the algorithm as noted previously. The EC transitions shown in Fig. 4 are associated with a Boolean condition that uses event input variables, input variables, output variables, or internal variables.

For a more detailed discussion of the Function Block Architecture, Brennan and Norrie [4] may be consulted.

3. Extending IEC 61499

In this section we primarily focus on the limitations of the IEC 61499 model with respect to achieving automatic configuration and reconfiguration of real-time distributed control systems and propose a general model that is built upon the IEC 61499 concepts to address this issue.

3.1. IEC 61499 and automatic configuration/reconfiguration

The IEC 61499 architecture described in the previous section primarily focuses on models and definitions that allow distributed applications to be developed using function blocks. In particular, IEC 61499 focuses on a precise definition of the function block that shares many of the characteristics of the traditional objects and agents used to develop distributed control applications (e.g., a traditional object focuses on data abstraction, encapsulation, modularity, and inheritance), but extends this concept (through the identification of data and event flow paths) to include process abstraction and synchronisation. As a result, this approach is particularly suitable for control of an environment that is concurrent, asynchronous and distributed.

In order to actually create and manage these systems however, further guidance is required. For example, the standard does not provide any guidelines concerning the actual implementation of function block applications: e.g., low-level communication protocol requirements (e.g., to load, initiate and configure function blocks), how function blocks are scheduled on resources, and how function block applications are compiled, downloaded and stored [24]. As well, intelligent control issues, such as how function block applications can be configured and reconfigured on-the-fly are understandably out of the scope of IEC 61499.

Over the past three years, the authors have been involved in various aspects of this work on distributed intelligent control. For example, in [3,40] we report on experiments with a distributed control operating system (DCOS) to support distributed function block applications. This prototype system was implemented on a distributed PC system consisting of Intel 486 and Pentium processors and was based on the notion of reactive agent architectures [13]. The results showed that our DCOS could support distributed function block applications and meet their basic requirements of concurrency, real-time event-based control. As well, various application configurations were compared (i.e., distributing function blocks across controller nodes in various ways), though the prototype did not support a means of dynamically reconfiguring these applications.

As noted previously, the research reported in this paper is focused on this issue. In particular, we focus on a model to support intelligent, dynamic reconfiguration that is based on IEC 61499 and can be implemented on a commercial real-time computing platform in the remainder of the paper. As will be discussed in

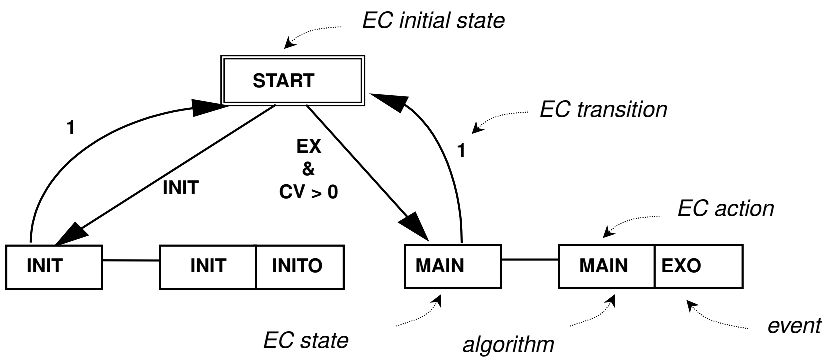


Fig. 4. An example of an IEC 61499 execution control chart.

the remaining sections, this model is intended to provide a consistent means of managing the configuration of IEC 61499 function block applications. Given that our distributed system is subject to hard and soft real-time constraints, and that it must be implemented on relatively small-footprint platforms typical for embedded control, it is implemented with light-weight, reactive agents (e.g. [13]). In order to provide the intelligent control capabilities that will make the system truly "holonic", this model will be interfaced with higher-level software agents (e.g., rational agents [32]), as described in the authors' work on reconfiguration methodologies (e.g. [5]), to provide the reasoning capabilities to achieve automatic configuration and reconfiguration.

3.2. *A conceptual model for configuration/reconfiguration*

Before describing our conceptual model for configuration control,[1] it should be noted that all of the descriptions of the models up to this point have been concerned with the execution of distributed control applications in a real-time environment. In particular, when considering IEC 61499 applications, the event and data flow paths determine the order of execution of the function blocks that describe a given distributed control application. By convention, this application execution control path is represented graphically as a horizontal flow path in IEC 61499 (typically from left to right, though feedback from right to left is permitted) as illustrated in Figs 5(a).

When considering configuration and reconfiguration of function block applications, the authors recognised that a second control path is required: i.e., a configuration control path. In order to distinguish this aspect of a real-time distributed control application from control application execution, we propose modelling configuration control as a vertical (top to bottom) control path as illustrated in Fig. 5(b) (in this case, the "configuration command" is concerned with adding a new function block "new FB"). The resulting conceptual model is shown in Fig. 5(c). In this figure, the IEC 61499 convention for control application execution flow is retained, while a new configuration control path is introduced.

As will be described in detail in the next section, two new agent types are introduced to help manage configuration. First, an Execution Agent (EA) is provided that, as its name implies, is primarily concerned with function block execution (e.g., implementing function block functionality, ensuring that function block algorithms are scheduled, etc.) but also plays the role of "monitor agent" in the configuration control path as is illustrated in Fig. 5(c). This monitoring capability allows the higher-level configuration management application discussed in the previous section to make decisions based on the current state of the execution agents. For example, execution agents can report on the application's deadline performance in order to allow the configuration management application decide whether or not function blocks must be redistributed (e.g., because a particular resource is not fast enough or the loading across resources is imbalanced). The second type of agent introduced in our model, the Configuration Agent (CA), is primarily responsible for implementing the configuration or reconfiguration plans generated by the configuration management application. For example, configuration agents are responsible for creating or deleting function blocks and maintaining inter-function block connections.

[1]The phrase "configuration control" is used to refer to the processes of configuration (e.g., during initial control application set up) and reconfiguration (e.g., when a control application is changed).

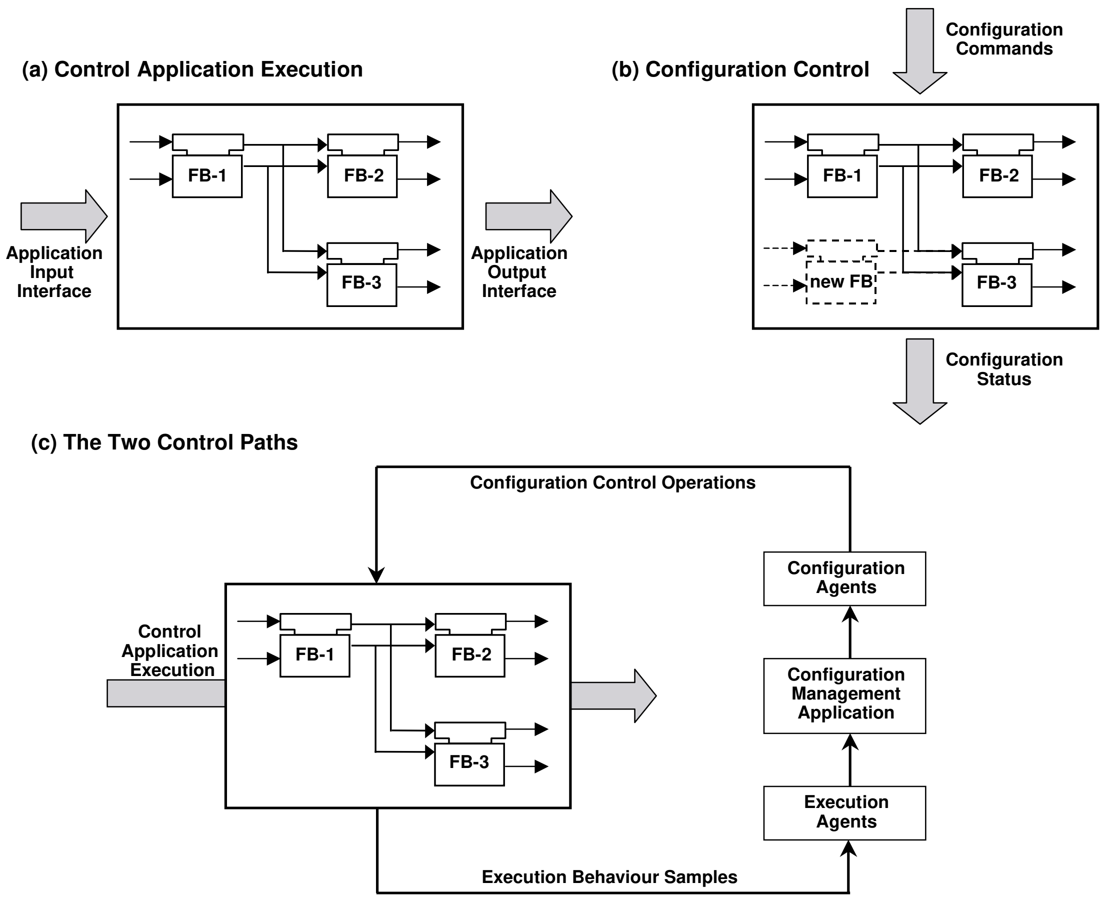

(a) Control Application Execution
Application Input Interface
FB-1
FB-2
FB-3
Application Output Interface
(b) Configuration Control
Configuration Commands
new FB
Configuration Status
(c) The Two Control Paths
Configuration Control Operations
Control Application Execution
Configuration Agents
Configuration Management Application
Execution Agents
Execution Behaviour Samples

Fig. 5. Conceptual model for configuration/reconfiguration.

In the next section, we look at this general model more closely, then focus on the details of this approach can be applied to the IEC 61499 basic function block.

4. The reconfiguration model

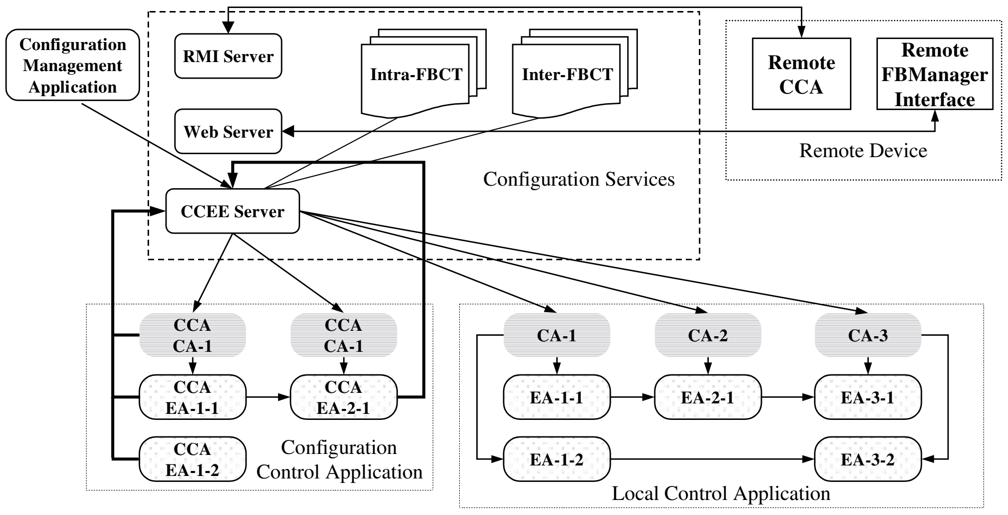
In this section we begin with a description of the model for configuration control, then describe how this model links with the IEC 61499 function block model.

4.1. Configuration control services and support architecture

The architecture for configuration control, shown in Fig. 6, consists of three basic modules to enable control application configuration and reconfiguration: (i) the previously discussed configuration management application, (ii) a configurations services module, and (iii) a configuration control application. As well, Fig. 6 shows the local control application and a remote device, where parts of the control application may be distributed or other control applications may be running (i.e., as described in Fig. 1). In the remainder of this sub-section, we will look at each of these aspects of the configuration control architecture.

The local control application, shown in the lower right of Fig. 6 is intended to represent an IEC 61499 function block application. As was noted previously (and will be described in greater detail in the next sub-section), IEC 61499 function blocks (e.g., Fig. 2) are modelled by configuration agents (CA) and execution agents (EA).

Similar to local control applications, the configuration control application (CCA) is also modeled by function blocks in our model. The main difference here is that the CCA is a special type of control application that executes a pre-determined configuration that is provided by the configuration execution engine (described next). Just as a control application controls the behaviour of physical devices (e.g., robots, CNC machines), the CCA control the behaviour of the control



CA – Configuration Agent
EA – Execution Agent
CCA – Configuration Control Application
CCEE – Configuration Control Execution Engine
FBCT – Function Block Connection Table

Fig. 6. Configuration control services and support architecture.

application. In other words, the CCA can be thought of as a meta-control application that is responsible, for example, for how function blocks in the local control application are interconnected.

The configuration services module shown in Fig. 6 acts as an interface between this meta-control application and the higher-level configuration management application, where configuration and reconfiguration plans are developed. As well, since control applications exist in a distributed environment, it also serves as an interface to other devices.

The key elements of the configuration services module are the configuration control execution engine (CCEE) and the connection tables (intra-function block connection table (intra-FBCT) and inter-function block connection table (inter-FBCT)). The CCEE basically completes the loop shown in Fig. 5(c): i.e., it monitors the status of execution agents (and reports this information to the configuration management application) and relays the configuration commands (from the configuration management application) to the configuration agents. To enable the CCEE to convert higher-level configuration commands from the configuration management application (e.g., “connect function block ADC1 to function block PID3 on RESOURCE2”), the CCEE relies on two sets of tables that describe the connections within function blocks (intra-FBCT) and the connections between function blocks (inter-FBCT).

Finally, configuration and reconfiguration can be managed across distributed devices either automatically (i.e., invoked by a remote device’s CCA) or manually through a Remote Function Block (FB) Manager Interface. The configuration services provided to support these two modes of configuration are remote method invocation (RMI) and web services respectively.

As noted previously, the key to achieving intelligent reconfiguration (i.e., multi-agent techniques that enable the system to reconfigure automatically in response to change) with this model lies in the configuration management application. A detailed discussion of this application is beyond the scope of this paper however, for the remainder of this section we described two approaches to configuration management that have been investigated by the authors (e.g. [5]): (i) a pre-programmed or “contingencies” approach, and (ii) a soft-wiring approach.

For example, with the first form of configuration control, contingencies are made for all possible changes that may occur. In other words, alternate configurations are pre-programmed based on the system designer’s understanding of the current configuration, possible faults that may occur, and possible means of recovery.

With this approach, the configuration management application uses pre-defined reconfiguration tables that make use of the intra-FBCT and the inter-FBCT tables described above. For example, in the event of a device failure, the affected portions of an application could be moved to different devices by selecting an appropriate reconfiguration table. As well, this detailed representation of the function block interconnections would allow higher-level agents to access the information required to make a smooth transition from one configuration to another, thus enabling dynamic reconfiguration.

The main disadvantage of the contingencies approach is that it is inflexible, particularly with respect to handling unanticipated changes. As well, this approach would require constant maintenance in order to keep the reconfiguration tables current: i.e., each change would require a change to the reconfiguration tables.

The basic idea behind the second approach to reconfiguration is to take advantage of higher-level reasoning agents to analyse the current configuration and plan for reconfiguration when required. Ideally, we are striving for an "integrated circuit" approach where fine grain components can be "plugged-in". Similar to Sun's Jini approach, this approach uses the directory services of the intra-FBCT and the inter-FBCT as well as the underlying configuration agents that handle the "wiring" between components.

The primary advantage of this approach is its potential to overcome the inflexibility of the contingencies approach as well as its potential to realise intelligent reconfiguration. For further details of these two approaches to reconfiguration please see [5].

4.2. *Configuration control and the function block model*

The IEC 61499 standard specifies a control component model at the logical level. It also defines general principles for the specification of types and the behaviour of instances of service interface function blocks, including management function blocks. As well, the command syntax for configuration control is specified. However, the standard does not specify models for function blocks to support runtime reconfigurability. Based on the requirements of a holonic control system noted previously, the authors have designed a multi-agent model for basic and composite function blocks to support runtime reconfigurability.

Given that function blocks, like holons, are recursive, the basic function block (BFB) model that is described in this sub-section is equally applicable to IEC 61499 composite function blocks and sub-applications [21].

To model the configuration process and component interaction for function blocks, the IEC 61499 model must be combined with other modelling approaches to specify and verify the execution process and reconfiguration process concurrently. Several models have been proposed in recent years to describe real-time embedded software [10]. Since many embedded systems deal with safety-critical applications, formal specification and verification are highly desired and widely used in software development, such as StateChart [17], Net Condition Event System [35], Coloured PetriNet [41], etc.

It is also recognised that a disadvantage of using formal methods to specify a system is that an executable system cannot be constructed after specifications are done, and the implementation can introduce additional errors even if the specifications have been proven to be correct [10].

To model IEC 61499 function block based control components with concurrent configuration control, we combine the function block model with the Hierarchical Finite State Machine (HFSM) model to generate a concurrency model from the software architecture point view as shown in Fig. 7.

In this figure, basic function blocks are modelled as follows. First, the function block's control functionality (i.e., how it is expected to behave in the control application) is encapsulated in execution agents, which have a direct correspondence with the basic function block model shown in Fig. 2. For example, the IEC 61499 ECC (execution control chart) is represented by the execution agent's basic function block ECC (BFB-ECC); the IEC 61499 function block algorithms and internal data are encapsulated in EA task agents. One form of representation for the execution agent that has been suggested by the authors [15] is to use the real-time unified modelling language (RT-UML) "capsule" stereotype (the symbols used for the BFB task agents in Fig. 7 correspond to the symbol for RT-UML capsules). This model allows state machines to be modelled explicitly and also supports a recursive structure.

As noted previously, configuration agents manage function block configuration. This is illustrated graphically in Fig. 7 by the BFB configuration agent. In order to manage which agent (EA or CA) is executing, or in other words, whether the function block is in the execution or configuration flow path (i.e., Fig. 5), an additional state machine is used: the basic function block configuration control chart (BFB-CCC). This state machine can be in one of three states: configuration ("C"), execution ("E"), or stopped ("S"). In the "configura-

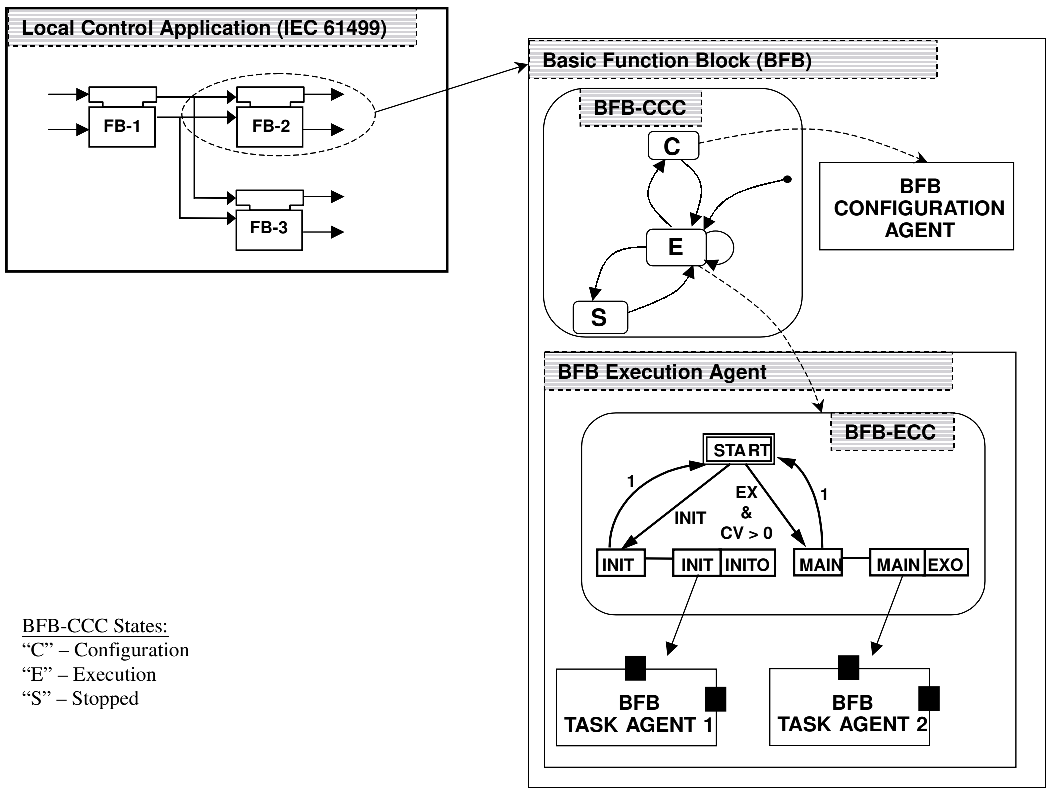


Fig. 7. Hierarchical Finite State Machine (HFSM) model.

tion" state, the BFB-CA is running and performing configuration commands that are based on messages received from the CCEE. In the "execution" state, the BFB-EA runs, allowing the function block to perform its control application functionality. Finally, the function block can be placed in a stopped state if required (e.g., prior to a configuration change).

Figure 8 illustrates how this approach fits into the IEC 61499 basic function block model. As noted in Section 2, IEC 61499 control flow runs from left to right by convention. This convention is shown by the event and data inputs and outputs flowing from left to right through the execution agent in Fig. 8. Like the standard IEC 61499 model shown in Fig. 2, event I/O is associated with the execution control chart (i.e., BFB-ECC in Fig. 8) and data I/O is associated with internal data and algorithms (i.e., encapsulated in the task agents in Fig. 8).

The main modification to the basic function block model is the introduction of event and data flow in the new configuration control flow path described previously. In direct parallel to our control execution flow path, configuration event inputs and outputs are associated with the configuration control execution control chart (i.e., BFB-CCC in Fig. 8), while configuration data inputs and outputs are associated with one or more configuration agent. The reason that both data and event I/O are used in the configuration control path is that configuration and reconfiguration of function blocks is managed by the configuration control application (CCA) described in the previous sub-section. As noted previously, this application is also a function block application.

5. Implementing the configuration control model

In this section we provide an example of how our model to support configuration control has been implemented in real-time Java. We start with a brief description of the application environment, and then provide an example of a common proportional-integral-derivative (PID) controller application.

5.1. The application environment

As noted previously, our earlier work on a real-time distributed control prototype involved developing

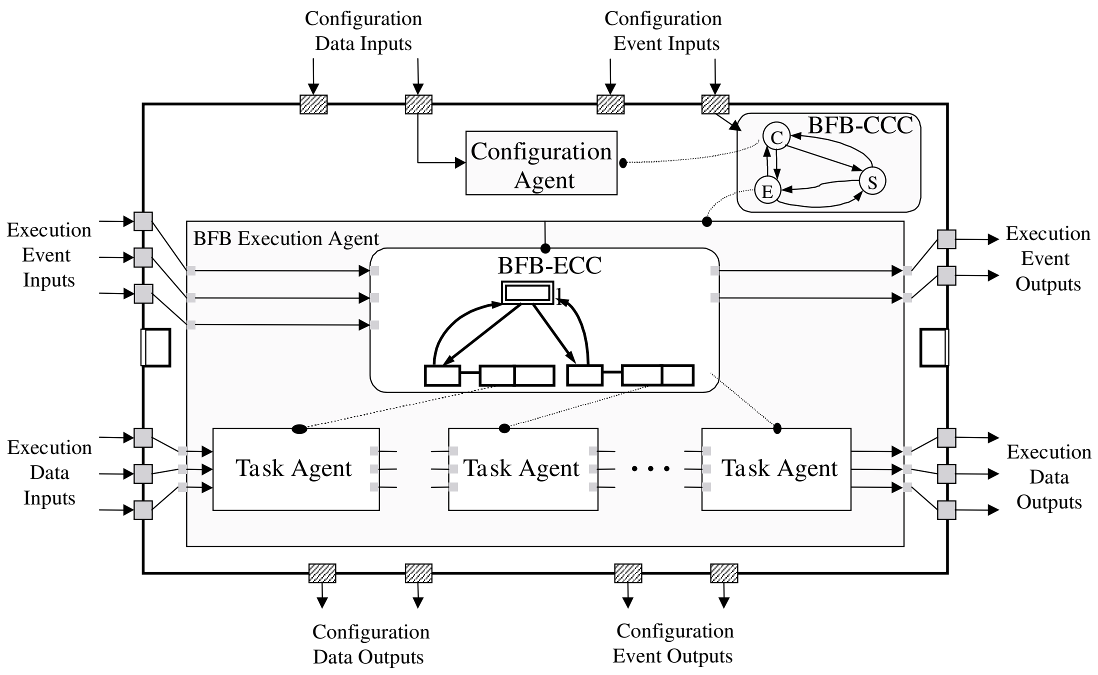


Fig. 8. The basic function block model.

a prototype real-time distributed operating system to support distributed IEC 61499 function block applications (e.g. [3,40]). Given the recent advances in both hardware and software for distributed applications, we chose to use a commercially available system to test our configuration control model.

For software development, we chose Java, because of its broad popularity, simplified object model, strong notions of safety and security, as well as its multithreading support. Of course, Java has not had a history of use in real-time embedded systems because of its large size, non-deterministic behaviour, and scheduling performance. Recently however, there has been considerable progress in the development of Java-based microprocessors [25] and Java real-time kernels [12] that have made real-time embedded Java-based system more of a reality.

Currently, there are three main approaches for real-time Java: (i) work towards a new Java specification of a real-time Java virtual machine [33], (ii) a Java chip for directly executing Java code with real-time performance [1], and (iii) combing the Java virtual machine with a real-time operating system kernel to supply an integrated real-time Java kernel [12]. For our work, we chose the third approach based of our experience with real-time operating systems and chose the Jbed operating system implemented on a single PowerPC 823e [20].

The Jbed operating system is designed to support the safe composition of components, objects, and tasks. This modular design allows components, such as the basic function block component described in Figure 8, to be loaded on demand. As well, full control over task execution is provided (e.g., multitasking scheduler, memory allocator, garbage collector, and exception handling control), to allow applications' real-time constraints to be met.

In order to develop real-time Java applications, a target byte-code compiler (TBCC) is provided. This flash compiler can download Java byte code from a network link, serial line, or EPROM and compiles it directly into RAM [12]. Flash compilers like Jbed's TBCC delay compilation, but only until load-time, not until run-time as just-in-time (JIT) compilers do. This avoids the speed and size overhead of an interpreter and the long latencies when calling some code for the first time. As a result, flash compilers provide the best solution for embedded real-time systems.

Finally, Jbed supplies a mechanism to use a hard real-time garbage collection algorithm that does not block time-critical tasks, i.e., it can be pre-empted anytime. This is important because unlike memory allocation, garbage collection is not under direct control of the programmer. The collector runs as a background thread with lower priority than all the time-critical tasks. The main goal of the Jbed memory manager was to make

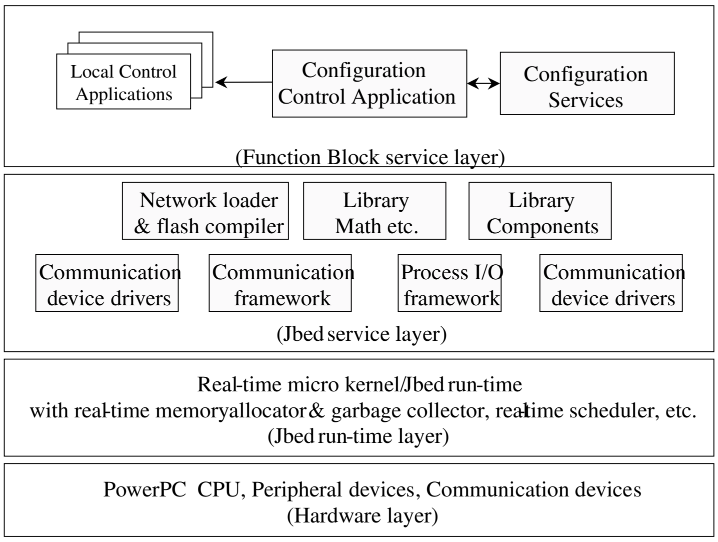
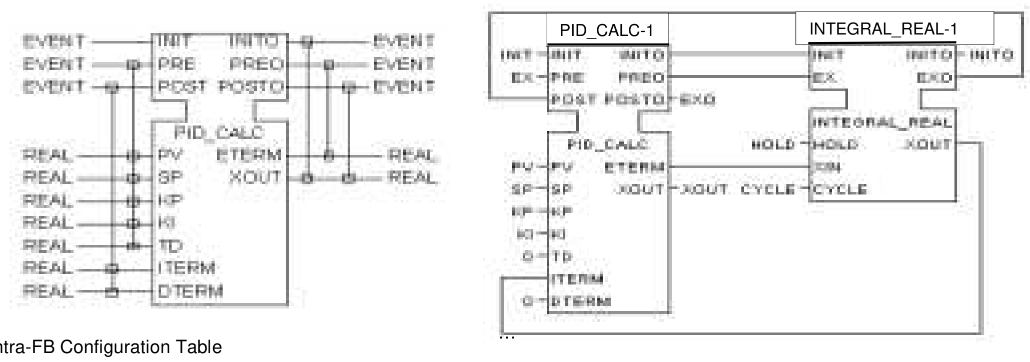

Local Control Applications
Configuration Control Application
Configuration Services
(Function Block service layer)
Network loader & flash compiler
Library Math etc.
Library Components
Communication device drivers
Communication framework
Process I/O framework
Communication device drivers
(Jbed service layer)
Real-time micro kernel/Jbed run-time with real-time memory allocator & garbage collector, real-time scheduler, etc.
(Jbed run-time layer)
PowerPC CPU, Peripheral devices, Communication devices
(Hardware layer)

Fig. 9. The Jbed-based implementation.

(a) PID_CALC Basic Function Block Type

(b) PID_CALC and INTEGRAL_REAL Function Block Application



(c) intra-FB Configuration Table

FB Name : (String)	PID_CALC-1	PID_CALC-1	PID_CALC-1	PID_CALC-1	PID_CALC-1	…
Execution Agent ID : (int)	2	3	4	Null	Null	…
Port Name : (String)	INIT	PRE	POST	ETERM	XOUT	…
Event ID : (int)	3	4	5	Null	Null	…
Data Type 1 Output Port ID : (int)	Null	Null	Null	1	2	…
Data Type 2 Output Port ID : (int)	Null	Null	Null	Null	Null	

(d) inter-FB Configuration Table

FB Name : (String)	PID_CALC-1	PID_CALC-1	PID_CALC-1	PID_CALC-1	…
Port Name : (String)	INITO	PREO	ETERM	Null	…
FB Name : (String)	INTEGRAL_REAL-1	INTEGRAL_REAL-1	INTEGRAL_REAL-1	INTEGRAL_REAL-1	…
Port Name : (String)	INIT	EX	XIN	Null	…

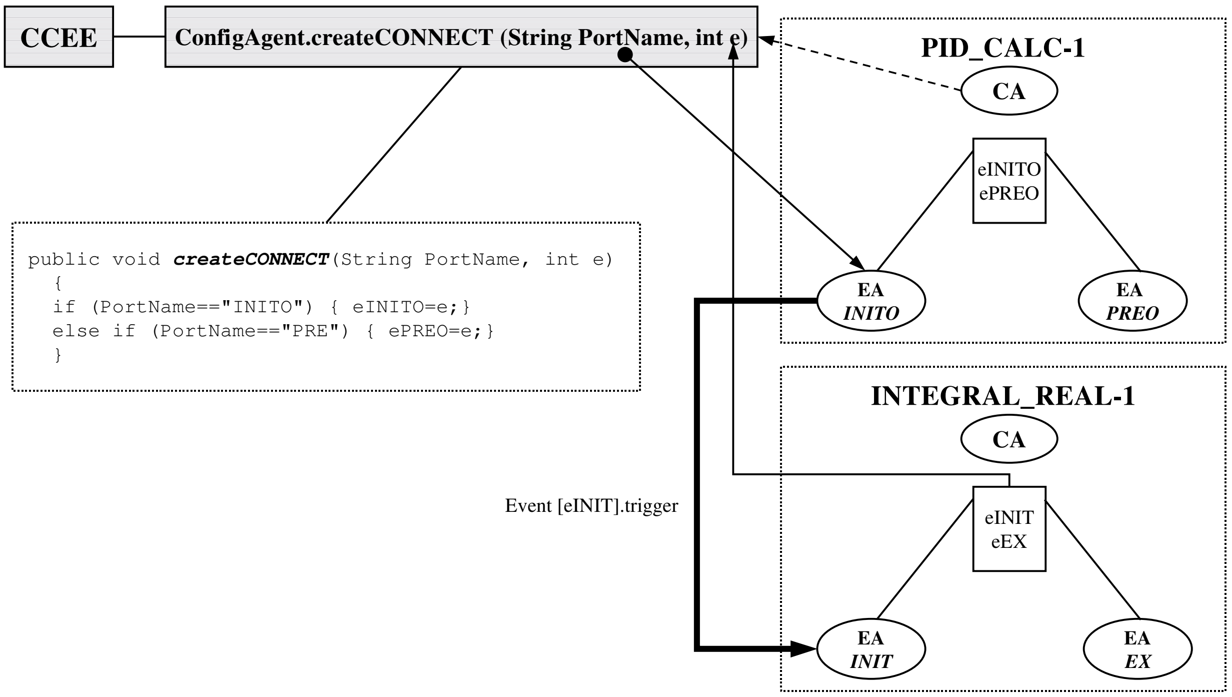
Fig. 10. A PID control application.

sure that garbage collection for non-time-critical tasks cannot interfere with time-critical tasks.

In order to test our approach, the Jbed-based Java development and execution environment is used as a platform to implement an experimental prototype of the configuration control model described in the previous section. Figure 9 provides an overview of Jbed-based controller structure to implement this experiment prototype. As can be seen in this figure, our configuration control model sits at the "application level" and is supported by basic Jbed services (i.e., compiling, class libraries, communication and I/O drivers), and the Jbed

```
int DURA=10; // Task Duration
int ALLO=0;   // Task Allowance
int DEAD=30; // Task Deadline

1    Get real-time event's id number
e=getEVENTid();

2    Create real-time event "e"
event[e]=new UserEvent();

3    Get execution agent's ID number
f=getFBid();

4    Create this real-time execution agent "f"
try { fbtask[f]=new Task(new PID_CALC_1("INIT"),DURA, ALLO, DEAD, event[e]); }
   catch (Exception e) { java.lang.System.out.print(e);}

5   insert port items into intra-FBCT
Set_intra_FBCT(getFBint("PID_CALC_1"), f, getPortint("INIT"), e, 0, 0, 0);

6   Start execution agent "f" (ready to be triggered)
fbtask[f].start();

// end of EA creation
```

Fig. 11. Execution agent creation.



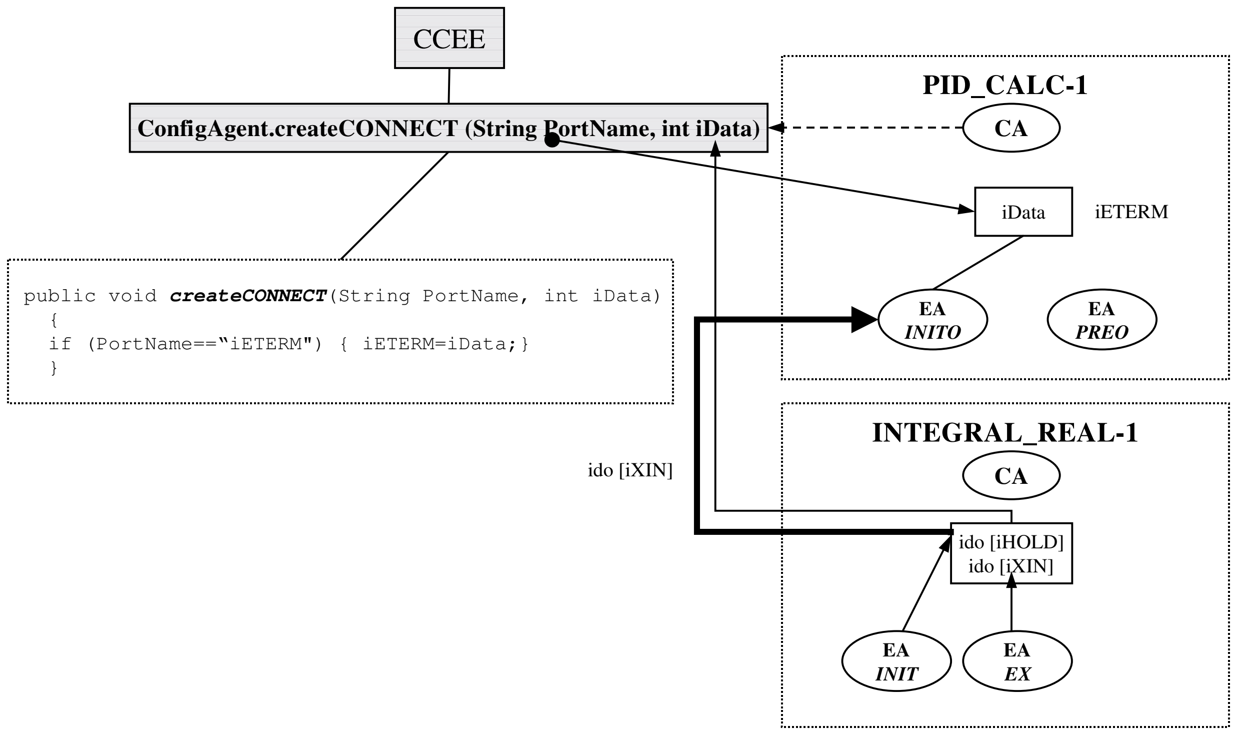
```
public void createCONNECT(String PortName, int e)
  {
  if (PortName=="INITO") { eINITO=e;}
  else if (PortName=="PRE") { ePREO=e;}
  }
```

Fig. 12. Inter-connecting function blocks (event connections).

real-time micro-kernel. As noted above, a PowerPC 823e is used for hardware support.

5.2. A worked example

In this section we provide an example of a simple PID control application that consists of two IEC 61499 basic function block types: PID_CALC (calculates the "proportional" and "derivative" terms) and INTEGRAL_REAL (calculates the "integral" term). Figure 10 shows the local control application (Fig. 10(b)) and also provides examples of the intra-FBCT (Fig. 1(c)) and the inter-FBCT (Fig. 1(d)).



```
public void createCONNECT(String PortName, int iData)
  {
  if (PortName=="iETERM") { iETERM=iData;}
  }
```

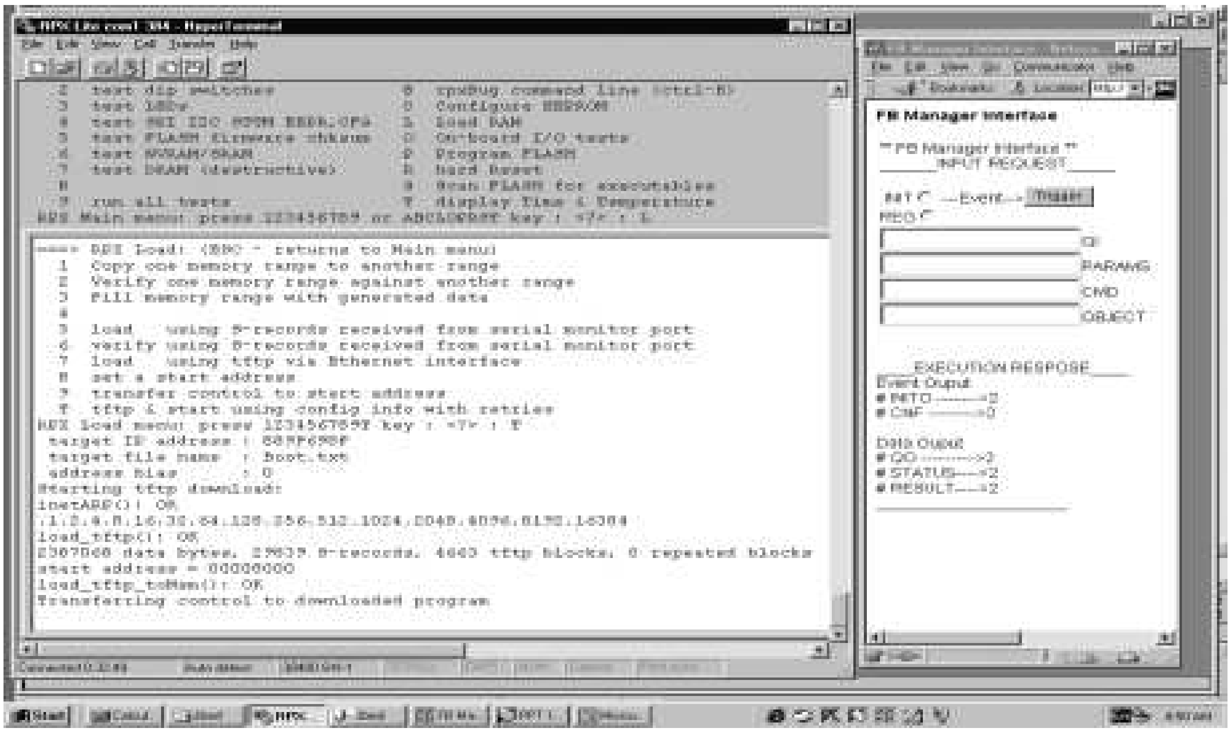
Fig. 13. Inter-connecting function blocks (data connections).

As noted previously, the intra-FBCT is concerned with the connections within the function blocks. In this case, Fig. 10(c) provides a detailed list of the connections within a specific instance of the PID_CALC function block type shown in Fig. 10(a) (i.e., instance "PID_CALC-1"). For example, the execution agent ID's and their associated ports are listed: i.e., "Port Name" and "Event ID" (in the case of event ports) and "Data Type" (in the case of data ports). Similar information is also provided concerning the task agent and configuration agent connections. As well, the table lists all function block instances in the local control application (i.e., in this case it lists "PID_CALC-1" and "INTEGRAL_REAL-1").

The inter-FBCT provides a listing of all of the connections between function blocks in the local control application. For example, the second column of Fig. 10(d) lists the event connection "INITO" (output of "PID_CALC-1") to "INIT" (input of "INTEGRAL_REAL-1"). In order to support run-time reconfigurability, these tables are programmed in an array structure and managed with a vector object structure. By maintaining these object structures centrally in the controller, the reconfiguration process is kept fast and simple.

In order to initialise these tables, the function block's configuration agent first creates the appropriate execution agents. The Java code for this process is shown in Fig. 11. Before describing the steps in this figure, it should be noted that, within each execution agent, task agents are associated with events. For example, in Fig. 7 when event "INIT" is true, state "INIT" is active and "Task Agent 1" runs. The direct parallel in the IEC 61499 function block model is illustrated in Fig. 4: i.e., when state "INIT" is reached, algorithm "INIT" executes. As a result, the first three steps of the execution agent creation process involve (1) getting a unique real-time event ID number for each function block event input (only one event is shown in this figure), (2) creating a real-time event for each function block event input, and (3) getting a unique execution agent ID number.

In the next step, when the execution agent is created, the events are assigned to task agents. The real-time task agent "fbtask[f]" is present in the object "Task" with real-time constraint parameters DURA (task duration), ALLO (task allowance), and DEAD (task deadline), and is associated with the real-time event object "event[e]". Other agents can now use the command, "event[e].trigger" to trigger "Task". We use the Java



```
  2   test dip switches              8   rpxBug command line (ctrl-B)
  3   test LEDs                      C   Configure EEPROM
  4   test SMI I2C MPMM EEPROM/OPs   L   Load RAM
  5   test FLASH firmware chksum     O   On-board I/O tests
  6   test NVRAM/SRAM                P   Program FLASH
  7   test DRAM (destructive)        R   Hard Reset
  8                                  S   Scan FLASH for executables
  9   run all tests                  T   display Time & Temperature
RPX Main menu: press 123456789 or ABCLOPRST key : <7> : L
===> RPX Load: (ESC - returns to Main menu)
  1   Copy one memory range to another range
  2   Verify one memory range against another range
  3   Fill memory range with generated data
  4
  5   load    using S-records received from serial monitor port
  6   verify using S-records received from serial monitor port
  7   load    using tftp via Ethernet interface
  8   set a start address
  9   transfer control to start address
  T   tftp & start using config info with retries
RPX Load menu: press 123456789T key : <7> : T
 target IP address : 88FF690F
 target file name  : boot.txt
 address bias      : 0
Starting tftp download:
inetARP(): OK
:1:2:4:8:16:32:64:128:256:512:1024:2048:4096:8192:16384
load_tftp(): OK
2307068 data bytes, 29837 S-records, 4443 tftp blocks, 0 repeated blocks
start address = 00000000
load_tftp_toMem(): OK
Transferring control to downloaded program
```

FB Manager Interface

** FB Manager Interface **
INPUT REQUEST

INIT ○ ---Event---> Trigger
REQ ○

QI
PARAMS
CMD
OBJECT

EXECUTION RESPONSE
Event Ouput
INITO ---------->2
CNF ---------->0

Data Ouput
QO ---------->2
STATUS----->2
RESULT----->2

Fig. 14. Function block manager interface.

reflection mechanism to facilitate run-time reconfiguration between "Task" and "event[e]".

As noted previously, internal function block event connections and data connections are maintained in the intra-FBCT. Step 5 of Fig. 11 inserts the new task agent and event object connection data in this example. Once all of the event objects and task agents are created and registered in the intra-FBCT, the execution agent is ready to run (i.e., step 6 of Fig. 11).

At this point, we have described the structure of the function block connection tables, inter-FBCT and intra-FBCT, as well as how individual function block instances are created and the intra-FBCT is updated. The final stage involves establishing connections between function block instances in order to create the local control application, as well as the associated updates of the inter-FBCT. Figures 12 and 13 show how function block event and data connections are established respectively. For example, in Fig. 12 a configuration agent establishes the event connection "INITO" (output of "PID_CALC-1") to "INIT" (input of "INTEGRAL_REAL-1") noted previously. In this case, PID_CALC-1's configuration agent creates a connection for its event port "INITO" (i.e., PortName = "INITO") and associates it with INTEGRAL_REAL-1's event port "INIT" and its real-time event "eINIT". Once this connection is established, an event at PID_CALC-1's "INITO" output event port will cause INTEGRAL_REAL-1's execution agent (EA) to perform whatever behaviour is associated with its event "eINIT" (e.g., an initialisation algorithm). Similarly, Fig. 13 shows the process for data connections. In this case, the connection is being made between PID_CALC-1's "ETERM" data output, and INTEGRAL_REAL-1's "XIN" data input.

At this stage the local control application is ready to run on the device (i.e., the execution agents have been created) and the configuration control services are ready to be used by a configuration application (e.g., a configuration management application as shown in Fig. 6). An example of this is provided in Fig. 14. In this case, we show the terminal interface to the PowerPC (running RPX Lite) and our Function Block Manager interface implemented on a web browser. The interface allows function blocks to be manually configured (i.e., both intra- and inter- function block connections) and the event and data connections to be tested. Our tests have shown that this approach allows small function block applications like the one described in this section to be quickly and easily configured and reconfigured at run time. Of course, to achieve a truly holonic system, as described in Section 1, further experimental work is required with larger applications and automatic reconfiguration mechanisms.

6. Conclusions and future research

In this paper we have described a concurrent function block model to control the run-time reconfiguration process of a real-time holonic controller. A key aspect of this model is its use of a hierarchical reconfiguration management that specifies two concurrent control paths: (i) control of the process, and (ii) configuration control (i.e., software concerned with managing application configuration and reconfiguration). As well, we described a real-time java implementation to support function block-based real-time task execution and run-time reconfigurability.

At the current stage of development, this approach provides a framework to support intelligent reconfiguration, but does not yet realise this holonic manufacturing systems goal of automatic and dynamic adaptability. In order to develop appropriate methodologies for reconfiguration, we have been following a sequential approach with our research [5], focused on increasingly "sophisticated" levels of reconfiguration: i.e., simple, dynamic and intelligent reconfiguration. For example, simple reconfiguration involves evaluating the utility of using the IEC 61499 model for reconfiguration (e.g., to avoid software coupling issues during reconfiguration). With dynamic reconfiguration, our focus has been on the development of techniques to properly synchronise software during reconfiguration. Tests with our prototype real-time distributed operating system [3,40] as well as our recent tests with the system reported in this paper have already made considerable progress towards these two forms of reconfiguration. Our recent work has been focused on the highest level of reconfiguration, intelligent reconfiguration, and in particular, how multi-agent techniques can be used to allow the system to reconfigure automatically in response to change.

References

[1] aJile Systems, Website, http://www.ajile.com/, 2002.

[2] A. Baker, H. Parunak and K. Erol, Agents and the internet: infrastructure for mass customisation, *IEEE Internet Computing* **3**(5) (1999), 62–69.

[3] S. Balasubramanian, R.W. Brennan and D.H. Norrie, An architecture for metamorphic control of holonic manufacturing systems, *Computers in Industry* **46**(1) (2001), 13–31.

[4] R.W. Brennan and D.H. Norrie, Agents, holons and function blocks: distributed intelligent control in manufacturing, *Journal of Applied Systems Studies Special Issue on Industrial Applications of Multi-Agent and Holonic Systems* **2**(1) (2001), 1–19.

[5] R.W. Brennan, M. Fletcher and D.H. Norrie, *An agent-based approach to reconfiguration of real-time distributed control systems,* submitted to: IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures, 2002.

[6] S. Bruckner, J. Wyns, P. Peeters and M. Kollingbaum, Designing agents for manufacturing process control, *Proceeding of Artificial Intelligence and Manufacturing Research Planning Workshop,* 1998, pp. 40–46.

[7] S. Bussmann and J. Sieverding, Specification of holonic agent control concepts in manufacturing logistics, Deliverable D7.3-1 of HMS Consortium, 2000.

[8] M. Cutkosky, R. Englemor, R. Fikes, T. Gruber, M. Genesereth, W. Mark, J. Tenenbaum and J. Weber, PACT: an experiment in integrating concurrent engineering systems, *IEEE Computer* **26**(1) (1993), 28–37.

[9] T. Darr and W. Birmingham, An attribute-space representation and algorithm for concurrent engineering, *AI EDAM* **10**(1) (1996), 21–35.

[10] B. Douglass, *Doing hard time: Developing real-time systems with UML, objects, frameworks, and patterns,* Addison-Wesley, 1999.

[11] C. Duarte and T. Maibaum, A rely-guarantee discipline for open distributed system design, *Information Processing Letters* **75** (2000), 55–63.

[12] Esmertec, *Jbed RTOS Package User Manual,* Esmertec, Inc., 2000.

[13] J. Ferber, Reactive distributed intelligence: principles and applications, in: *Foundations of Distributed Artificial Intelligence,* G. O'Hare and N. Jennings, eds, Wiley Interscience, 1996, pp. 287–314.

[14] M. Fletcher and R.W. Brennan, Designing holonic manufacturing systems using the IEC 61499 (function block) architecture, *IEICE/IEEE Joint Special Issue on Autonomous Decentralized Systems and Systems' Assurance* **E84-D**(10) (2001), 1398–1401.

[15] M. Fletcher, R.W. Brennan and D.H. Norrie, Design and evaluation of real-time distributed manufacturing control systems using UML Capsules, *7th International Conference on Object-oriented Information Systems,* 2001, pp. 382–386.

[16] M. Fox, J. Chionglo and M. Barbuceanu, *The integrated supply chain management system,* Internal Report, Department of Industrial Engineering, University of Toronto, 1993.

[17] D. Harel, Statecharts: a visual formalism for complex systems, *Scientific Computer Programming* **8** (1987), 231–274.

[18] T. Heikkila, M. Jarviluoma and T. Juntunen, Holonic control for manufacturing systems: design of a manufacturing robot cell, *Integrated Computer-Aided Engineering* **4** (1997), 202–218.

[19] Holonic Manufacturing Systems Consortium, Holonic manufacturing systems overview, Holonic Manufacturing Systems Consortium Web Site, http://hms.ifw.unihannover.de/, 2002.

[20] J. Hoskins and J. Blackledge, *Exploring the PowerPC Revolution,* Maximum Press, 1995.

[21] International Electrotechnical Commission, IEC TC65/WG6, Voting Draft: Function Blocks for Industrial Process-Measurement and Control Systems, Part 1 Architecture, International Electrotechnical Commission, 2000.

[22] K. Lam, G. Law and V. Lee, Priority and deadline assignment to triggered transactions in distributed real-time databases, *Systems and Software* **51** (2000), 57–65.

[23] R. Lewis, *Programming Industrial Control Systems Using IEC 1131-3,* IEE, London, 1996.

[24] R. Lewis, *Modelling Control Systems Using IEC 61499: Applying Function Blocks to Distributed Systems,* IEE, London, 2001.

[25] D. Loomis, *The TINI Specification and Developer's Guide,* Addison-Wesley, 2001.

[26] A. Lyons, *UML for real-time overview,* technical report of ObjecTime Ltd., 1998.

[27] F. Maturana and D.H. Norrie, Multi-agent mediator architecture for distributed manufacturing, *Journal of Intelligent Manufacturing* **7** (1996), 257–270.

[28] H. Park, J. Tenenbaum and R. Dove, Agile infrastructure for manufacturing systems: a vision for transforming the US manufacturing base, Defense Manufacturing Conference, 1993.

[29] H. Parunak, Manufacturing experience with the contract net, in: *Distributed Artificial Intelligence,* M.N. Huhns, ed., Pittman, 1987, pp. 285–310.

[30] H. Parunak, *RAPPID project index page,* http://www.iti.org/cec/rappid/, 1997.

[31] Y. Peng, T. Finin, Y. Labrou, B. Chu, J. Long, W. Tolone and A. Boughannam, A multi-agent system for enterprise integration, *Proceedings of PAAM'98*, 1998, pp. 533–548.

[32] A. Rao and M. Georgeff, An abstract architecture for rational agents, *Proceedings of Knowledge Representation and Reasoning,* 1992, pp. 439–449.

[33] The Real Time Java Expert Group, *The Real-time Specification for Java,* Addison-Wesley, 2000.

[34] J. Sieverding, Specification of manufacturing test missions, Deliverable D7.5-2 of HMS Consortium, 2000.

[35] R.S. Sreenivas and B.H. Krogh, On condition/event systems with discrete state realizations, *Discrete Event Dynamic Systems: Theory and Applications* **2**(1) (1991), 209–236.

[36] P. Valckenaers and H. Van Brussel, IMS TC5: holonic manufacturing systems technical overview, Holonic Manufacturing Systems Consortium Web Site, http://hms.ifw.uni-hannover.de/, 1994.

[37] H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts and P. Peeters, Reference architecture for holonic manufacturing systems: PROSA, *Computers in Industry* **37** (1998), 255–274.

[38] V. Vyatkin and H. Hanisch, Bringing the model-based verification of distributed control systems to the engineering practice, VI IFAC Workshop on Intelligent Manufacturing Systems (IMS'2001), 2001, pp. 152–157.

[39] L. Wang, R.W. Brennan, S. Balasubramanian and D.H. Norrie, Realizing holonic control with function blocks, *Integrated Computer-Aided Engineering* **8**(1) (2001), 81–93.

[40] X. Zhang, S. Balasubramanian, R.W. Brennan and D.H. Norrie, Design and implementation of a real-time holonic control system, *Information Science Special Issue on Computational Intelligence for Manufacturing* **27**(1–2) (2000), 23–44.

[41] B. Zhou, L. Wang and D.H. Norrie, Design of distributed real-time control agents for intelligent manufacturing systems, *Proceedings of the 2nd International Workshop on Intelligent Manufacturing Systems,* 1999, pp. 237–244.