Research article

A software framework for coping with heterogeneity in the shopfloor

Juan-Antonio Fernández-Madrigal, Cipriano Galindo, Ana Cruz-Martín and Javier González
System Engineering and Automation Department, University of Málaga, Málaga, Spain

Abstract

Purpose – The CIM framework pursues the integration of components in a manufacturing enterprise by means of computer systems. This, however, may be obstructed due to heterogeneity in the field: programmable controllers, robots, sensors and actuators, etc. in communications: different kinds of networks and/or field buses; and in the programming tools for all these devices. Thus a solution is needed to integrate heterogeneous software/hardware components in a well-defined and flexible fashion. This paper seeks to address these issues.

Design/methodology/approach – This paper proposes a metalanguage, called H, and a set of tools that serve for designing, implementing, deploying, and debugging distributed heterogeneous software on the shopfloor. The metalanguange includes fault-tolerance and real-time mechanisms, among other features.

Findings – The use of a framework that can integrate different software and hardware components enables the engineer to take advantage of the best features of each existing technology. The use of object-oriented techniques, concurrent and distributed programming, and the isolation of heterogeneous parts, have also important benefits in the reusability and optimality of the solutions.

Practical implications – The use of a metalanguage like H, that separates the parts of the application that depend on particular (heterogeneous) components from the parts that are portable, has, as a main implication, important improvements in the development time, effort, and cost of CIM projects.

Originality/value – H is the first metalanguage coping with heterogeneity through the complete development cycle of software for manufacturing applications. It also provides a formal and well-defined framework for future extensions.

Keywords Computer aided manufacturing, Manufacturing industries, Fault tolerance

Paper type Research paper

Introduction

The CIM framework pursues the management of the total manufacturing enterprise with the aid of computer systems, in particular is relevant the use of software to integrate hardware and software components in an application. Now-a-days, it is commonly accepted that enterprise integration can be divided into three levels: physical systems integration, application integration, and business integration. According to Clements (1997), physical systems integration copes with data exchange among physical systems connected through a network, application integration allows the management of systems independently of where information resides, and finally business integration deals with top-level monitoring and control of processes.

However, this integration effort is often obstructed due to a common and evident characteristic of modern production systems: heterogeneity. Heterogeneity appears in different areas within any factory. To begin with, a wide variety of equipment can be found on the shopfloor: programmable controllers, robots, AGVs, sensors and actuators, CNC machinery, warehouses and material handling systems, and so on. Also, communications inside the shopfloor and through the whole enterprise may involve different kinds of networks and/or fieldbuses. Finally, many of the elements must be programmed, either with commercial-off-the-shelf (COTS) solutions or in more general purpose languages (C, C++ Java...). Moreover, the programming skills of the staff will change depending on the complexity of their tasks, and all the components of an application are subjected to changes as technology evolves.

Software is a good starting point to cope with all this heterogeneity. Programming languages are very close to the root of the problem (since many components are programmable) and the plasticity of software makes it well situated to address it. However, approaching heterogeneity by forcing the use of a single software paradigm or language for integrating all the components is difficult to put in practice due to the need of the equipment manufacturers to adhere to it. It seems more appropriate to look for a solution that is able to cope with existing developments. Also, this would produce

The current issue and full text archive of this journal is available at www.emeraldinsight.com/0144-5154.htm

Assembly Automation
27/4 (2007) 333–342
 [ISSN 0144-5154]
[DOI 10.1108/01445150710827113]

This work was supported by the Spanish Government under research contract DPI2005-01391.

better results than a common paradigm when confronted to different needs and situations.

There are a number of CIM solutions for integrating enterprise elements. Some of them, like CIMOSA, PERA, GRAI, GERAM or ATHENA, deal with the integration in the three aforementioned CIM levels. The CIM Open System Architecture, or CIMOSA (1999, 2007) is an ESPRIT supported framework that focuses specially on business integration. PERA (2007), or Purdue Enterprise Reference Architecture developed at Purdue University, provides a model for the enterprise life cycle that has been applied to paper mill, oil seeds processing, polyethylene plant industries, etc. PERA claims that any enterprise can be vertically separated into three different aspects (facilities, people, and control and information systems), that can in turn be horizontally divided into a set of eight phases in order to get a valid model of the organization. The GRAI methodology (Doumeingts and Vallespir, 1995) is a modeling technique that provides a conceptual structure for representing the enterprise, a formalism to express such conceptual description, and tools to implement it. Generalised Enterprise Reference Architecture and Methodology (GERAM, 1999) is a framework defined by IFAC/IFIP Task Force on Architectures for Enterprise Integration. It has been constructed after analyzing some other architectural approaches, and it is intended to be a generalized architecture definition that can be applied through the whole enterprise life cycle. Actually, CIMOSA, GRAI and PERA are enterprise models that verify the GERAM specification. Finally, a different approach, Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their applications (ATHENA, 2004), is a European commission supported project which pursues that "by 2010, enterprises will be able to seamlessly interoperate with others". ATHENA mainly studies the interoperability of applications, but also takes into account business processes and the contexts.

From a lower level perspective, there exist CIM solutions that cope with the physical and application CIM integration levels, for example, G++ (Aarsten *et al.*, 1995), SEMATECH CIM (1998), and OSEFA (Schmid, 1995, 1996). They focus on the integration of a wide variety of hardware, keeping in mind that this must be done by allowing production flexibility. Though, the approach presented in this paper also fits in this perspective, ours is an open solution that also considers heterogeneity in software (unlike G++ which uses only the JAVA and C++ programming languages and is a commercial solution), being not tied to the MES philosophy of SEMATECH. It is not only oriented to manufacturing cells, as OSEFA.

There are other CIM solutions, nowadays mostly concerned with robotic devices, related to the physical integration level. Most of them are non-commercial frameworks that facilitate the implementation of robotic/machinery software. However, it must be said that, except for OSACA, they are mainly aimed to research. Player/Stage (Collett *et al.*, 2005) is a good example of an open source software tool that enables the control of mobile robots and sensor devices, though it cannot be considered a complete framework since the user must entirely develop the client-side software. OROCOS (Flanders, 2005) provides a free-cost, suitable set of pre-built libraries for machine tools and robotic arm control. Some other open approaches are OSACA in Europe, OSEC in Japan, and OMAC in the USA (Pritschow *et al.*, 2001), although none of them has been widely adopted by industry. Our previous works, called NEXUS (Fernández-Madrigal and González, 1999) and BABEL (Fernández-Madrigal *et al.*, 2007b), from which the solution presented in this paper is derived, have demonstrated a higher level of flexibility than these solutions in terms of lifecycle phases covered, automatic generation of code, programming languages, communications support, etc.

In this paper, we propose a new approach to cope with heterogeneity and flexibility from the programming point of view, aimed to produce software applications that adapt as best as possible to the heterogeneous components of the shopfloor. We claim that our approach covers both the physical and the application CIM integration levels, since interoperatibility among different elements and/or applications on the shopfloor is achieved by means of a common and transparent information exchange methodology. As mentioned before, our approach is an evolution of previous works on the same direction, but with a completely new formalization and features for fault-tolerance and communication frameworks. It consists of a simple and well defined metalanguage, called H, for designing and implementing distributed software modules (programs in charge of controlling mechanisms) that can communicate through different classes of networks. Each of these modules, in turn, can define its functionality with a different programming language in order to adapt to heterogeneity in software. The H metalanguage includes concurrence, fault-tolerance and real-time features to specify the different requirements of the modules and hardware devices that compose the application.

We have also provided H with a set of tools that cover all the stages of the development lifecycle of the application. We have been testing preliminary versions of the H and the H-tools in the BABEL approach for robotic applications in our automation research labs during the last ten years. In this paper, we illustrate the new formalization with a case of study of a CIM shopfloor application. The obtained results have been promising, providing the engineer with a framework that reduces the cost of development substantially while coping with heterogeneous hardware and software with reduced effort.

A comparative between our H metalanguage and tools and some of the CIM solutions previously mentioned is offered in Table I. This comparison covers the physical and application CIM integration levels; the business level is excluded since deals with high-level issues (decision making, simulations, etc.) that, at this moment, are out of the scope of our methodology.

The plan of the paper is as follows. In the next two sections, we introduce the H metalanguage, describe the H tools, and explicitly enumerate the real-time and fault-tolerance characteristics of our approach. Then we present a case of study on the shopfloor. Finally, some conclusions and an outline of some future work in this area are provided.

The H metalanguage and the H tools

In this section, we outline our specification metalanguage for the design and programming of heterogeneous applications, called H (for "Heterogeneity"), and a set of tools that produce, deploy, validate, and debug executable programs

Table I Comparative table between H language and other relevant CIM solutions

Framework	Lifecycle phases covered	Automatic generation of code	OS limitations	Programming language	Communication platform	Real-time	Integration levels
OROCOS	Implementation, testing	No	Linux. Complete implementation of the framework for other OSs	C++	CAN, CORBA in progress	RTAI	Physical
OSACA	Implementation, testing	No	Win32, VxWorks. Complete implementation of the framework for other OSs	C++	TCP/IP	None	Physical
G++	Implementation, testing	Yes	Microsoft Windows NT 4.0/2000/XP/95/98/ME, OSF-Motif for some UNIX environments	C++ JAVA	CORBA-like and MS D-COM-like	Yes	Physical, application
SEMATECH	Design	No	NA	NA	NA	NA	Physical, application
OSEFA	Design	No	NA	NA	NA	NA	Physical, application
H	Design, implementation, testing, maintenance	Semiautomatic	Win32, LynxOS, JAVA VM, and no-OS. Only few modifications for further OSs	C, C++ JAVA, STEP-5. Only few modifications for further languages	ACE + TAO CORBA, TCP/IP, USB, and monolithic. Only few modifications for further platforms	LynxOS RT and RTX support. Only few modifications for further platforms	Physical, application

Notes: NA – not applicable entries are related to some aspects of CIM solutions that are mainly oriented to design issues, and therefore do not explicitly consider implementation features

from those designs. For a more technical and detailed specification of both you can consult (Fernández-Madrigal *et al.*, 2007a). H has been conceived for producing codifications, even in different programming languages, of distributed software with real-time and fault-tolerance requirements. It has the following main features:

- heterogeneity enabled;
- distributed programming;
- concurrent programming;
- real-time;
- fault-tolerant;
- extensible; and
- validation/debugging enabled.

In H, an application is composed of a set of modules, being each module executed on a single computational machine and providing a number of public services to other modules. During the design, the components of the application and its requirements are separated conveniently into the heterogeneous components and the portable ones. The whole design can be divided into the following steps:

1 structural design of modules;
2 codification design of modules (programming);
3 enumeration of heterogeneous hardware/software components that support the execution of the codifications of step 2;
4 structural design of the application; and
5 definition of an implementation for the application.

Heterogeneity does not appear in step 1, which covers only the public structure of modules. This allows the engineer to reuse as much as possible these structures (since they do not depend on any particular hardware or software). Examples of structural designs for a CIM application can be seen in the case of study section. One of the new features of H with respect to our previous works is the inclusion of inheritance in the design of the structure of modules, which improves even more reusability by enabling the ordered modification of existing modules, the composition of simpler designs into more complex ones, and the definition of structures that are intended only for derivation and not for instantiating any codification (abstract modules).

A second feature intended for improving reusability is the use of a data specification language aimed to write type definitions and service parameters in the structural design. This language is called HDL (for "Heterogeneous Data Language"), and it is a subset of the OMG's CORBA and IDL (2007a, b). All the structural designs use the same language for defining data.

Finally, in the structural design, some real-time features can be included, in particular the priority of services with respect to other services in the same module. During the deployment of the application, a priority is assigned to each module; thus, the final absolute real-time priorities of services depend on both specifications.

Step 2 is the first place where heterogeneity appears, although it does weakly. Each codification is associated to the structural design of a module (it "implements" that module). For improving reusability and facilitating versioning, H includes a new feature, inheritance, which allows the engineer to vary existing codifications to produce new ones (for example, because they are to be executed on a different hardware).

Module codifications follow an active object paradigm (Brugali and Fayad, 2002), that is, from their perspective a module is a program that contains an internal status and provide some services (concurrently executed) and data

definitions to other modules. A codification includes the programming sequences for each service and its real-time characteristics (execution time). The services' logics are written in some programming language, which is the main reason why we call H a "metalanguage": it allows the designer to cope efficiently with the existing heterogeneity in programming. Figure 3 shows examples of codifications.

For keeping the codification of modules as portable and reusable as possible (and heterogeneity as weak as possible at that level), the programmer can use what we call "H atoms": macros that can be included within the codification of services for carrying out some non-portable operations, for example, requesting a service from another module. In Figure 3, some examples of H atoms (in red) are shown. Although H atoms cover a number of necessities, some portions of code may be still dependent on a particular hardware/software component (for example, a software library for mathematical processing or a data acquisition card). These components must be explicitly listed in a special section associated to the code, called deportabilization. This encourages the programmer to make heterogeneity explicit at the codification level.

Step 3 is devoted to enumerate the particular hardware and software components involved in an application. This is where heterogeneity is most widely coped with. We classify heterogeneous components in general platforms: hardware, execution, communications, real-time, and fault-tolerance. Particular instances of these classes (called Particular Platforms) represent the actual heterogeneous components involved in the application, for example, a given operating system or a concrete processing hardware. There is no limit in the number of particular platforms that can be defined, which is the main reason why our approach is highly extensible.

In step 4, an application structure is defined that includes a number of modules (not codifications, since it is a structural view of the application). The possibility of including more than one copy of a given module is a new feature of H with respect to our previous works: it allows the engineer to have more than one identical device where the module will run. An example of application structure written in H is shown in Figure 4.

The last step (step 5) serves for defining an implementation, that is, an instantiation of the structure of an application, in order to produce executable programs. An example of implementation in H is shown in Figure 4. Several implementations of the same application structure can be provided, for example, if there are changes in the distribution of the modules among the computers, or in the hardware support.

In the implementation design, there must be specified the support relations existing between particular platforms. For example, a given computer can provide support for a given operating system, which in turn can provide support for a given processing library.

The optional fault-tolerance section is a new feature of H. It allows the engineer to activate fault-tolerance mechanisms through replication for each module, either active or passive. A deployment section is intended to distribute the codifications of the modules among the available execution/hardware platforms. If a given codification is deployed more than once (several replicas), H has provision for the following cases of fault-tolerance:

- In passive replication, a replica number will define the order in which the replicas will be selected for processing requests. When the highest number ceases to respond after a given timeout, the next one will take its place. Whenever, the selected replica produces the result, its internal status will be replicated in all of the others.
- In active replication, the replica number will be used to index the output data produced by each replica. These results will be merged/coordinated by the latest replica in finishing the request.

Once all the steps of design have ended, the executable programs that compose the implementation of the application can be produced automatically. For that purpose, the H tools include a software called H-apc, for "Heterogeneous Application Constructor" that interprets the designs described previously and generates source code for the specified codification languages, execution/hardware platforms (operating systems or CPUs), and a number of compilers and interpreters. Currently we have a version of this tool integrated into our previous visual CASE application of the BABEL framework, called the BABEL module designer. It includes support for codifications written in C, C++ JAVA, and for execution platforms that include MS Windows NT + LynxOS, and JAVA VM. In the future we plan to extend the list of platforms supported, which can be efficiently accomplished due to the internal structure of the H-apc, widely based on scripts.

For deploying and executing a given application, once it has been constructed with H-apc and compiled/linked if needed, we use another software, called H-apx (for Heterogeneous APlication eXecutor), that is able to launch the programs on their respective computational machines from a remote station, and collect their logging information when that option is activated. The list of deployments indicated in the implementation also configures the relative priorities of the codifications and their launching order.

If logging has been activated for some codification, the results can be analyzed off-line through our H-apl (Heterogeneous Application Logger), a software that shows graphically the sequence of events that have occurred during execution. This allows the engineers to inspect possible real-time flaws or bad ordering of requests, apart from user-defined events that can be registered at any time.

Finally, all the designs produced with H can be stored in our BABEL web site (BABEL Homepage, 2007) for versioning and maintenance. The site includes multi-user privileged accesses, version support, and some validation tools. For example, it is possible to examine a set of modules to detect loops in their service requests or the impossibility of satisfying the real-time requirements specified in the timing keywords of codifications.

Real-time and fault-tolerance

In this section, we describe in more detail both the real-time and fault-tolerance mechanisms enabled by H, since both are relevant issues in CIM and process control.

H enables to integrate both hard real-time and soft or non-real-time software into the same application through the use of a set of optional features. For example, for a SCADA system the monitoring software could provide a less stringent real-time performance than the shopfloor control devices. Thus, H allows the designer to include real-time requirements as needed, both at module and service granularities. Obviously, these requirements should only be specified if the underlying execution and hardware platforms enumerated

in the specification of the implementation of the application support them (operating system, virtual machines, CPUs, etc.). All real-time facilities described in the following are included in the real-time class of particular platforms (examples of these platforms are POSIX 1003.4 (Gallmeister, 1995), VenturCOM RTX (Ardence, 2005), and many others, thus we capture heterogeneity in real-time services):

- Hard real-time priorities for modules and services, in a hierarchical fashion. Each service within a module may be assigned a priority number relative to other services (there is no bound on this priority numbers). Each module, in turn, may be assigned a priority number relative to other modules. The H-apx application is in charge of mapping this priority specification to actual priorities of the execution platforms (operating systems) that support the modules, assuring the monotony of the selected priorities, although not strictly[1]. In the case that such support is not available (for example, a module implemented on an on-line interpreter with no real-time performance), evidently the design of the application will not fit the intended requirements.
- *Time requirements specification.* In each service, an estimate of its WCET (worst case of execution time) and BCET (best case) can be provided. On the other hand, in each service request specified with an H atom, a desired requirement for the execution time of the request can be set. This allows the designer to check out by simple validation tools if the overall real-time requirements of the application can be satisfied.
- *Real-time scheduling and synchronization.* An application specified in H is inherently concurrent (and may be also distributed). Services are the minimal unit of sequential execution, thus they are tasks. For specifying their scheduling, services can be of two basic types: reentrant and non-reentrant. The former are those that can be requested (and thus, executed) concurrently without other limitations, thus they need synchronization mechanisms (H provides some H atoms for synchronization, based on the semaphore metaphor). The latter are services that only are served when no other service is running. The supporting software and hardware for guaranteeing these simple requirements must be provided by some real-time particular platform (an operating system, a virtual machine, or a hardware such as a CPU).

In addition to these real-time specifications, H has provision for the following cases of fault-tolerance:

- *Passive replication.* In the deployment of an implementation of an application, a replica number will define the order in which the replicas of a codification will be selected for processing requests. When the highest number replica ceases to respond after a given timeout, the next one will take its place. Whenever, the selected replica produces the result, its internal status will be replicated in all of the others.
- *Active replication.* In this case, the replica number will be used to index the output data produced by each replica. Their results will be merged/coordinated by the latest replica in finishing the request. No communication is needed to replicate the status since all the replicas process the same data; however a broadcast mechanism is needed to send the request to all of them. Notice that active replication is suitable for modules that do not request services from other modules (or from theirselves), since in other case all the replicas will issue the request, generating a non proper operation. For modules that request services, it is more suitable to use passive replication.

The fault-tolerance section of the implementation of an application must include some fault-tolerance particular platform for the replicated module, in order to provide the necessary support for the replicas, mainly broadcast facilities and result-merging algorithms.

A case of study

Figure 1 is a case of study of a simple industrial cell in charge of classifying goods that are transported on a conveyor. Its desired operation is as follows: a good on the belt is transported until a presence sensor detects it, which triggers the visual inspection of the object. The recognition system is composed of two intelligent cameras that merge their individual results in order to improve the robustness of inspection. They capture an image and class the object according to its brightness. The result of the classification makes the routing mechanism (a diverter on the belt) to head the object to the correct direction.

In the following, we describe in more detail the hardware and software available in this plant and the design and implementation of a control and SCADA application using H and the H tools. It will be shown that, in spite of the apparent simplicity of the plant, the heterogeneity can be quite high.

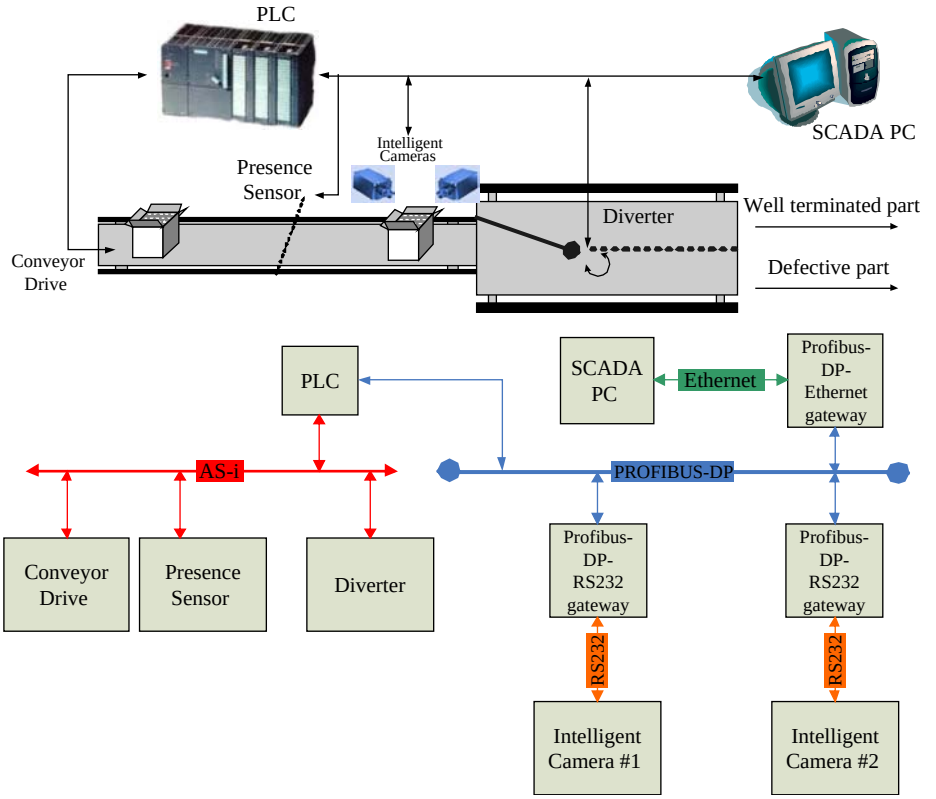
Hardware and software available

At the field level, the plant has the following sensors, actuators, and controllers:

- A presence sensor (typically a photoelectric sensor (Pepperl-Funchs, 2007)) that gives an on/off binary output under the presence or not of an object on the belt.
- A conveyor drive, composed of a gearmotor and a starter (MOVIMOT, 2007) attached to the pulley of the conveyor belt. It has no reverse operation and no speed regulation.
- A pneumatic diverter (Kuhnke Airbox, 2007) that admits an on/off signal for selecting one out of two lanes in the last segment of the belt.
- Two intelligent B/W cameras with embedded processors with capturing and basic image processing capabilities (Siemens, 2001). Each one provides a two bits output indicating the presence of a well terminated part (00), a defective part (11), or a not-known (01) or error status (10).
- A PLC controller. For this case study we will consider an out-of-catalog device in order to augment heterogeneity and also illustrate the typical reuse of components in many manufacturing environments. In our example it is the modular Simatic S5-100U controller (Siemens, 2002).

These devices are connected through a heterogeneous network. Non-programmable sensors and actuators (presence, drive, diverter) are connected through an AS-i bus to the PLC (through an AS-i module (AS-i, 2007)). The controller is connected to the cameras as a master in a Profibus-DP bus (Profibus, 2002) (a CP 5431 module for this PLC is needed (Siemens, 2000)). This hard real-time bus will also be used for connecting the field to the SCADA system. Finally, the VS710 cameras chosen for this case of study provide direct connection to Profibus-DP, but for increasing

Figure 1 A case study of a shopfloor application. Top: Illustrative scheme of the process. Bottom: heterogeneous elements involved in the application and their network connections



heterogeneity, we will consider their RS-232 connections instead. So, we need a pair of Profibus-DP/RS-232 gateways (Shauf, 2007) that set the cameras as slaves in the Profibus.

The enterprise level will contain a number of conventional PCs. One of them will include a SCADA software. We will consider that all these PCs share an Ethernet network, and the SCADA computer will pass through to the field by means of a Ethernet/Profibus-DP gateway (Applicom, 2007), being seen as a master from the Profibus perspective.

Concerning the software, there are three programmable devices in our example:

1 Both intelligent cameras include an Intel 486 processor and run MS-DOS 6.22 (Microsoft, 2007). The development is carried out on a different computer using Visual C++6.0 (Kruglinski, 1997) if C or C++ languages are chosen, and the binary files transferred to the embedded processors via RS-232 or the other available connections.
2 The PLC has a ROM with a cyclic executive. The programs can be written in AWL (Siemens, 2002), which is a textual programming language similar to assembler. They are developed in an external computer and transferred to the PLC via a serial cable. Once there, they can be stored in an EPROM.
3 The PC in charge of running the SCADA front-end will run a MS-Windows XP OS (Microsoft, 2007), which can be programmed in a variety of languages. This computer will serve both as a development and as an execution environment. We will choose JAVA (Sun, 2007) for programming our front-end due to its graphical capabilities and portability among platforms.

Designing the application with H

In the design of this application with the H language, we can distinguish three different modules, intended for managing each of the programmable devices: a PLC Control module, a SCADA front-end module and an inspection module. Figure 2 shows some example of the structural designs for these modules. Notice the use of inheritance. The PLC Control module has some hard-real time constraints in the form of a cyclic task in charge of sequencing the shop plant. Also notice that the SCADA Front-End receives information from the plant asynchronously, without hard time requirements (it is also able to act on the plant to stop or start its operation under user demand). No dependency on a given hardware or software is needed yet, and thus the portability and reusability of these structural designs is maximum.

The next step is to design the codifications for these structures. This is when heterogeneity must be dealt with for the first time. On the one hand, specific programming languages must be selected. We will choose JAVA for the SCADA Front-End, C for the embedded camera processors, and AWL for the Simatic PLC. On the other hand, some information on the particular platforms that are needed for executing the codifications may be included. Notice that through the use of H atoms, most of the portability/heterogeneity parts of the codification will still remain portable. Figure 3 shows some fragments of the

Figure 2 Structural designs of the fielddevice and controldevice abstract modules, and of the PLCControl module which inherits from the controldevice

```
Module abstract structural design FieldDevice
    Description: "An abstract module that encapsulates a general field device"
    Author: "The H Team"
    Data definitions
        enum DeviceStatus {ok,stopped,failure};
    End data definitions
    Service GetStatus
        Characteristics: reentrant;
        Priority: dynamic
        Outputs: DeviceStatus st;
        Description: "Return the current status of the device"
    End service GetStatus
End module structural design FieldDevice
```

```
Module abstract structural design ControlDevice inherits from FieldDevice
    Description: "An abstract module that encapsulates a general control field device"
    Author: "The H Team"
    Data definitions
        enum DeviceStatus {ok,stopped,failure};
    End data definitions
    Service NumberOfDevices
        Characteristics: reentrant;
        Priority: dynamic
        Outputs: long num;
        Description: "Return the number of devices controlled by this"
    End Service NumberOfDevices
    Service GetStatusOfDevice
        Characteristics: reentrant;
        Priority: dynamic
        Inputs: long numdev; // Index of device to check, from 0
        Outputs: bool invaliddev, FieldDevice::DeviceStatus st;
        Description: "Return the current status of the indicated device"
    End service GetStatusOfDevice
    Service GetStatusOfDevices
        Characteristics: reentrant;
        Priority: dynamic
        Outputs: sequence<FieldDevice::DeviceStatus>;
        Description: "Return the current status of all the devices controlled by
                      this, including itself at the end of the sequence"
    End service GetStatusOfDevices
    Service SuspendControl
        Priority: prioritized;
        Description: "Suspend the execution of the system controlled by this"
    End service SuspendControl
    Service ResumeControl
        Priority: prioritized;
        Description: "Resume execution of the system if it is in suspension"
    End service ResumeControl
End module structural design FieldDevice
```

```
Module structural design PLCControl inherits from ControlDevice
    Description: "Control of the sensors and actuators of a case of study shop floor application"
    Author: "The H Team"
    Service MainControl
        Characteristics: monitor, permanent absolute 500 milliseconds;
        Priority: prioritized
        Description: "This service is a cyclic hard-real time task in charge of checking
                      the status of the presence sensor and stop/start the conveyor drive
                      and the diverter as necessary after consulting the Inspection module"
    End service MainControl
End module structural design FieldControl
```

codifications for the modules. The Inspection module includes active replication for merging the results of both cameras. The PLC main control service is deportabilized by the CP 2433 module (the one that provides access to the AS-i bus from the PLC), since it is used to access field devices that do not contain modules.

In addition to, the design of modules and codifications, the particular platforms involved in the application must be enumerated for an implementation to be produced. We give a table with the platforms of our example in Table II.

The application can now be designed by simply listing the modules involved in the application (PLCControl, Inspection, and SCADAFrontEnd) possible implementation is shown in Figure 4.

Implementation, execution, debugging, and maintainance with the H tools

From the designs proposed previously the H tools can produce implementations. First of all, the implementation file (Figure 4) serves as an index to collect all the modules and codifications involved in the application. The H-apc tool is intended to produce the source code of complete programs from the codifications, including code for communications and support of the H atoms. The source code produced from each codification must be compiled/linked by some existing software related to the codification language and to the execution/hardware platform where the module's codification will be run. For example, for the Inspection modules' codifications, the source code must be compiled/linked by MS Visual C++6.0 for constructing a console application suitable for MS-DOS 6.22; for the SCADA Front-End, the JAVA compiler must be executed to produce a JAVA bytecode program. Once the codifications have been transformed into executable, independent programs, they must be transferred to the corresponding execution/hardware platforms.

The second H tool to use is the H-apx. It is in charge of launching the codifications by following the sequence order specified in the implementation. Our current version would be able to do this from a centralized computer in a network using TCP/IP communications. Further, extensions are needed to include remote launching in PLCs.

Once the application is terminated (which can be commanded also from the H-apx), the debugging results can be collected and passed to the H-apl.

Figure 3 Codification designs of the PLCControlS5 and inspection modules

```
Module codification design PLCControlS5 implements PLCControl
    Description: "Codification of the PLC Control module for executing in a Simatic S5 PLC"
    Author: "The H Team"
    Codification language: step-5-awl
    Internal status
        {-{
                DB2
                   KY = 1   #state of the system 1 -> ok, 0->failure, 2 -> stopped#
                BE
        }-}
    End internal status
    Startup logic // This logic will be dumped to the OB 21 block of the S5 by the H tools
        {-{
                    S M 1.0
                    S M 1.1
                    R M 2.0
                    R M 2.1
                        ...
                    = A 0.1    #start moving the conveyor#
        }-}
    End startup logic
    Service MainControl
    // This logic will be dumped to OB 1, which is executed cyclically by the S5
        {-{
                    C DB 2
                    ...
                    #calling the inspection service. Results are stored in two marks#
                    #_H-atom(Inspection,Inspection,"M 1.0,M 1.1")_#
                    U M 1.0
                    U M 1.1
                    = A 0.2   #it moves the diversion device to 1 if the result is 11, or 0 otherwise#
                    O(
                    U M 1.0
                    UN M 1.1
                    )
                    O(
                    UN M 1.1
                    U M 1.0
                    )           #result is 01 or 10, that is, a failure in the system#
                    T BI 1
                    = A 0.1     #the conveyor is activated if system is ok#
                    ...
                #The rest of the code#
            }-}
        Deportabilization: SimaticPLC // Since the codification cannot access the non-programmable devices
without it
    End service MainControl
    // The other services written here…
End module codification design PLCControlS5
```

```
Module codification design VS710CameraInspector implements Inspection
    Description: "Codification of the inspection logic for the VS710 camera"
    Author: "The H Team"
    Codification language: ansi-c
    Service Inspect, timing 10 milliseconds .. 100 milliseconds
      {-{
        // Accesses to the camera
        ...
        // Dump image into main memory
        ...
        // Run recognition process
        ...
      }-}
      Replication:
         {-{
            int res1=result[0]; // result of the service as produced by the first replica
            int res2=result[1]; // the result produced by the second replica
            if (result[0]==result[1])
                result=result[0]; // both replicas have returned the same
            else
            {
                // If one of the replicas does not work, take the other
                if (result[0]==Inspection::error)
                    result=result[1];
                else if (result[1]==Inspection::error)
                    result=result[0];
                else // both replicas disagree; take the worst case
                    result=Inspection::defective;
            }
         }-}
    End service MainControl

    // The other services written here
End module codification design VS710CameraInspector
```

Finally, all the files involved in the application can be stored in our BABEL web site (currently they are compacted into a single file) for maintenance. This site also provides facilities to check some dependency problems between modules and the satisfaction of real-time requirements.

Conclusions and future work

We have presented in this paper a meta-programming framework for coping efficiently with heterogeneity in the shop-floor (as long as the shopfloor includes programmable devices). We have developed a number of code generators for H in the last years under the previous BABEL framework, and this can be extended with an unbounded number of generators in the future. Basically, the H tool that is most affected by the inclusion of new particular platforms and codification languages is the application constructor H-apc. Thus, we have developed it based on several templates that can be easily changed and added to the tool, in most cases without recompiling it. Thus, heterogeneity in the future is guaranteed to be coped with properly.

Table II List of particular platforms considered in the case of study

Particular platform	General platform	Description
SimaticPLC	Hardware platform	The Simatic S5-100U (CPU 103) PLC basic hardware and operating system. Also the CP2433_ASi module for accessing the AS-i devices
SimaticPLCTiming	Real-time platform	Hardware and software support for the real-time features of the simatic PLC
CP5431_Profibus	Communications platform	Simatic module for communications through Profibus-DP
SimaticProfibus_Broadcast	Fault-tolerance platform	A software library for performing broadcast operations from the Simatic-Profibus side, needed in the active replication of the intelligent cameras
VS710Camera	Hardware platform	The hardware of the VS710 intelligent camera, including the processor, memory, and internal connection to the CCD sensor
MSDOS622	Execution platform	The operating system for the VS710 camera embedded processor, including the software libraries for accessing the camera
RS232_MSDOS622	Communication platform	A software communication platform for issue service and event requests through an RS232 connection on a MS-DOS 6.22 OS
PC	Hardware platform	The conventional PC in charge of the SCADA front-end
JAVA_VM	Execution platform	The JAVA Virtual Machine for executing the front-end
EthernetProfibusGateway	Communication platform	Hardware and software platforms for providing support for Profibus communications through ethernet gateway

Figure 4 The application file and a particular implementation of the case of study

```
Implementation OurImplementation for CaseStudyApplication
    Description: "An implementation that uses the codifications given previously"
    Author: "The H Team"
    Platforms:
        AnOldS5 is SimaticPLC,
        S5Timing is SimaticPLCTiming,
        AProfibusModule is CP5431_Profibus,
        BuiltinBroadcast is SimaticProfibus_Broadcast,
        Camera1, Camera2 are VS710Camera,
        Copy1OfDOS, Copy2OfDOS are MSDOS622,
        CRS232Library1, CRS232Library2 are RS232_MSDOS622,
        EnterprisePC1 is PC,
        OurCopyOfJAVAVM is JAVA_VM,
        EPGateway is EthernetProfibusGateway
    Support:
        AnOldS5 supports S5Timing, AProfibusModule, BuiltinBroadcast;
        Camera1 supports Copy1OfDOS, CRS232Library1;
        Camera2 supports Copy2OfDOS, CRS232Library2;
        EnterprisePC1 supports OurCopyOfJAVAVM, EPGateway;
    Fault-tolerance:
        Inspection uses BuiltinBroadcast for active replication;
    Deployment:
        PLCControlS5 deployed on AnOldS5 order 2 pause user logging on;
        JAVASCADAFrontEnd deployed on OurCopyOfJAVAVM order 3 pause user;
        VS710CameraInspector replica 0 deployed on Camera1 order 1;
        VS710CameraInspector replica 1 deployed on Camera2 order 0;
End application CaseStudyApplication

Application CaseStudyApplication
    Description: "Design of the application
from a structural point of view"
    Author: "The H Team"
    Modules: PLCControl, Inspection,
SCADAFrontEnd
End application CaseStudyApplication
```

Some features that are not dealt with currently in the H metalanguage itself and may be subjected to further work in the future are: migration mechanisms for codifications, higher levels of specification/design (for example, layers over H that are specialized in particular domains: manufacturing, robotics, etc.) more complex and mathematically grounded validation mechanisms, non-textual codification languages (for example, Statecharts (Harel, 1987), reflexive properties to handle the structure of the application from within the application, automatic re-launching of failed codifications with resuming of previous internal status, more sophisticated asynchronous communications between modules, exception handling at the design level, etc.

Note

1 Several specified priorities in H could be mapped into the same actual priority of an operating system in the case that the OS does not have enough support for real-time performance, such as the MS Windows family of OSes.

References

Aarsten, A., Elia, G. and Menga, G. (1995), "G++: a pattern language for the object oriented design of concurrent and distributed information systems, with applications to computer integrated manufacturing", *Pattern Languages of Program Design, Software Patterns Series*, Addison-Wesley, Reading, MA.

Applicom (2007), APP-ESP-GTW applicom® GATEway profibus to Ethernet or serial, available at: http://woodhead.com

Ardence (2005), available at: www.ardence.com

AS-i (2007), available at: www.as-interface.net/

ATHENA (2004), ATHENA European Integrated Project, available at: www.athena-ip.org/

BABEL Homepage (2007), available at: www.babel.isa.uma.es/babel/

Brugali, D. and Fayad, M.E. (2002), "Distributed computing in robotics and automation", *IEEE Trans. On Robotics & Automation*, Vol. 18 No. 4.

CIMOSA (1999), CIM Open System Architecture, available at http://cimosa.cnt.pl/

CIMOSA (2007), Enterprise Engineering and Integration, available at: www.cimosa.de/

Clements, P. (1997), "Standards support for the virtual enterprise", *Assembly Automation*, Vol. 17 No. 4, pp. 307-14.

Collett, T.H.J., MacDonald, B.A. and Gerkey, B.P. (2005), "Player 2.0: toward a practical robot programming framework", paper presented at Australasian Conference on Robotics and Automation (ACRA'05), Sydney.

Doumeingts, G. and Vallespir, B. (1995), "A methodology supporting design and implementation of CIM systems including economic evaluation", in Brandimarte, P. and Villa, A. (Eds), *Optimization Models and Concepts in Production Management*, Gordon and Breach Science Publishers, New York, NY, pp. 307-31.

Fernández-Madrigal, J.A. and González, J. (1999), "The NEXUS open system for integrating robotic software", *Robotics & Computer-Integrated Manufacturing*, Vol. 15 No. 6.

Fernández-Madrigal, J.A., Galindo, C., Cruz-Martín, A. and González, J. (2007a), "The H metalanguage and the H tools", Technical Report (Draft), Dpto de Ingeniería de Sistemas y Automática, Universidad de Málaga, May 2007.

Fernández-Madrigal, J.A., Galindo, C., González, J., Cruz-Martín, E. and Cruz-Martín, A. (2007b), "A software engineering approach for the development of heterogeneous robotic applications", *Robotics & Computer-Integrated Manufacturing*, (to appear).

Flanders (2005), Flanders Mechatronics Technology Centre, The OROCOS Project, available at: www.orocos.org

Gallmeister, B. (1995), *POSIX 4. Programming for the Real World*, O'Reilly and Associates, Sebastopol, CA, ISBM 1-56592-074-0.

GERAM (1999), available at: www.cit.gu.edu.au/ ~ bernus/taskforce/geram/versions/geram1-6-3/v1.6.3.html

Harel, D. (1987), "StateCharts. A visual formalism for complex systems", *Science of Computer Programming*, Vol. 8, pp. 1-4.

Kruglinski, D.J. (1997), *Inside Visual C++*, Microsoft, Lake City, FL.

Kuhnke Airbox (2007), available at: www.kuhnkeairbox.com/index.php

Microsoft (2007), available at: www.microsoft.com

MOVIMOT (2007), available at: www.sew-eurodrive.com

OMG CORBA (2007a), *Schmidt D. ACE + TAO Corba Homepage*, available at: www.cs.wustl.edu/ ~ idt

OMG IDL (2007b), available at: www.omg.org/

Pepperl-Funchs (2007), "Diffusive sensor with background suppression", DataSheet RL28-8-H-400-RT-B3B/73c. Pepperl + Fuchs, available at: www.am.pepperl-fuchs.com/pdf/documents/rl28-8-h-400-rt-b3b-73c-datasheet.pdf

PERA (2007), available at: http://pera.net/

Pritschow, G., Altintas, Y., Jovane, F., Koren, Y., Mitsuishi, M., Takata, S., Van Brusel, H., Weck, M. and Yamazaki, K. (2001), "Open controller architecture – past, present and future", *CIRP Annals*, Vol. 50, pp. 463-70.

Profibus (2002), *System Description*, Version October 2002, Order Number 4.002.

Schmid, H.A. (1995), "Creating the architecture of a manufacturing framework by design patterns", paper presented at Tenth annual conference on Object-oriented programming systems, languages, and applications, Austin, TX, pp. 370-84.

Schmid, H.A. (1996), "Creating applications from components: a manufacturing framework design", *IEEE Software*, Vol. 13 No. 6, pp. 67-75.

SEMATECH (1998), *Computer Integrated Manufacturing (CIM) Framework Specification Version 2.0*, SEMATECH, Inc., Austin, TX, January.

Shauf (2007), PROFIBUS – RS232 Gateway DSPI_RS, SHAUF, available at: www.schauf.haan.de

Siemens (2000), Automation & Drives Product Type AS-Interface Master for SIMATIC S5 Order No. 6GK1 243-3SA00.

Siemens (2001), SIMATIC VS 710. Quick Reference Guide A5E00032597-02 edition 11/2001.

Siemens (2002), S5-100U. Programmable Controller. System Manual. EWA 4NEB 812 6120-02a.

Sun (2007), Sun's JAVA homepage, available at: www.sun.com/java/

Corresponding author

Cipriano Galindo can be contacted at: cipriano@ctima.uma.es