ADVENT OF GENERIC COMPONENTS IN A MECHATRONIC CONTROL SOFTWARE ENVIRONMENT

M. SINGH, A. WHITE, R. GILL, S. PRIOR
Engineering Systems Group
Middlesex University
Bounds Green Road, London N11 2NQ
UNITED KINGDOM

Abstract: - Integrating multidisciplinary areas in a mechatronic environment means using information technology to provide appropriate access to accurate and timely information. It has become almost impossible for the heterogeneous systems, which provide solution to various problems accurately, to share the information and also use the resources of other systems. Integration of various hardware devices and software applications has made the integration of complex applications and mechatronic devices difficult. This paper describes an approach for minimising the effort of integration and maximising the reusability and efficiency of software for mechatronic systems by the creation of generic components. These generic frameworks may be used with other control systems.

Key-Words: - Generic Component; Open Control Systems; Reusability; Modularity; Interoperability; Mechatronics

1 Introduction

In an integrated manufacturing environment, the task of creating and maintaining applications, which drive mechatronic devices, has been difficult. A variety of programming languages has been used to generate applications by software engineers, resulting in inflexible, difficult to change application codes. Almost every time such applications required a completely new development cycle to adapt them to a new mechatronic device. Also due to the pace of development in technology supporting such devices, the software driving them seems to have degraded with time, and cannot be 'fixed'. Software engineers who develop the systems subsequently become unavailable and the problem is left in the hands of manufacturing engineers. They are left with the jargon of codes that need maintenance and lack an easy way for making changes. To best serve the interests of the manufacturing engineers, the application software tools should be tailored to their knowledge base. Manufacturing engineers are experts in designing and operating manufacturing systems. They know how the environment operates and understand the logic. Yet, they cannot program the control and integration logic required for manufacturing applications because they may not trained to use particular programming languages. Instead, industry usually depends upon software engineers to handle programming and integration requirements. It is a highly iterative and tedious process in which manufacturing engineers typically conceptualise the system, define the specifications, and spell out requirements while the application programmers attempt to code the application accordingly [1]. More often, software engineers tend to develop applications based on their own interpretation of what works best.

2 Background and Approach

The mechatronic devices are themselves very flexible, in contrast to specific control systems that are inherently inflexible. The application developers for compatibility have to use the proprietary programming environment and network protocols. Usually in a manufacturing environment where such devices are used, the information is scattered across the environments, and has incompatible networks, devices, computers and operating systems. This results, in loss of productivity and applications are flooded with large amount of data on single system resulting in poor performance.

The concept of a software control environment refers to a collection of tools used by developers to produce software for mechatronic systems. The tools composing most of such applications are often

thought out, designed, and implemented separately. This raises the following deficiencies:

- Redundancy: The same functionality may be supplied by several tools within the same control environment.
- Lack of integration: Such control environments do not provide facilities for new tools design and existing tools integration.
- Lack of interworking: The heterogeneity of tool interfaces makes communication difficult between tools.
- Lack of openness: There is no support for co-operation between tools belonging to different control environments.

In summary, the major bottlenecks for implementing integrated systems are device and application integration. Incompatibility of devices hampers portability and information exchange.

To remove these deficiencies and to respond to an increasing requirement for device integration support, an attempt is made to define the generic components, which supports integration of mechatronic devices. A generic component is an integrated program to execute certain functions in mechatronic systems. Integration is the term used to describe interworking between different devices. It is also defined as the ability to combine and reuse the funtionalities offered in an environment in flexible ways. According to Thomas and Nejmeh the capabilities relevant to integration are [2]:

Composition provides means to seamlessly combine arbitrary functions with no knowledge of precise tools or implementation in the environment.

Cooperation encourages tools to share information and resources.

Collaboration enables joint work through distributed and parallel tasks.

Coordination ensures the coherence and consistency of the composite, cooperative, and collaborative aspects of the systems.

These considerations led to the definition of generic components: a means for specifying uniform tools interfaces based on the concept of service signature; a support for service combination based on the notion of service scripts; and anonymous service access protocol based on the tools-interfaces import/export model.

Further, the design of generic components should take account the distributed and heterogeneous nature of current interconnected mechatronic systems. The generic components should be based on following characteristics:

- Location transparency: For generic components interworking must be supported while ensuring location independence. This means that mechatronic control applications that embed generic components use functions and resources supplied by remote generic components without knowing the location of that component on network.
- Openness: Generic components must support communication while blinding that the resource is from different component.

3 Object-Oriented Mechatronic Device

Although, mechatronic systems have their own architecture and implementation, almost all mechatronic systems are based on similar principles and functions. On careful and comprehensive analysis of the architecture of a general mechatronic device, it can be divided into several fundamental functional generic components. As mentioned in section 2 generic components are integrated programs, which are specified by abstracting the common characteristics.

Elements involved in implementation of mechatronic device can be divided into hardware devices, operating system and software for the application. Clearly, the software for application can take advantage of a common integration as opposed by the operating system and hardware devices that are rigid.

3.1 Development of generic components

Fig.1. shows the development cycle of the generic components. A possible domain is chosen and the model for that domain is developed. As mentioned in section 3, the division of the domain, in our case a mechatronic system is based on the abstraction of common characteristics. The result of this comprehensive and careful division is the realisation of behaviour, in an object-oriented manner. It is to be noted that the behaviour of so called generic components is actually inherited from the functional level abstraction of the software environment. The functional level contains the components, which implement control loops, behaviours, and general, basic processes, which are not concerned with the decision level of software environment. The characterisation of these generic components is tested against the actual application and feedback is provided for any deficiencies. After the successful realisation of the generic components required, they are specified using their parameters and attributes.

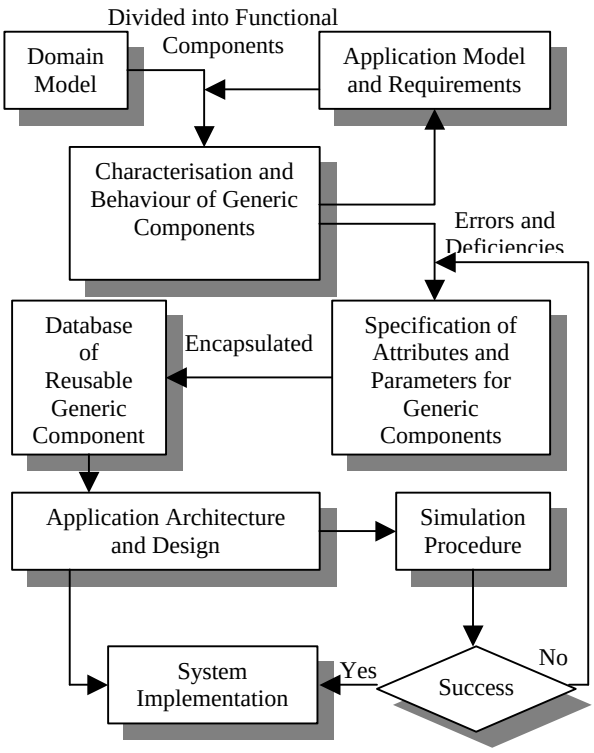

Fig.1. Development cycle of generic components.

The functionality is encapsulated and the outcome is a database of reusable generic components that are complete in themselves but cannot be implemented unless they are made available to the resources provided by the similar components. The application architecture is designed for simulation, if the simulation is successful the implementation is provided.

3.2 Generic components in mechatronic system

The main objective behind the development of generic components is to formalise the fundamental components as independent reusable entities and to standardise their input and output parameters as simply as possible and then to encapsulate their functionality. It is quite clear that the system should be based on object oriented techniques, where the resource required is accessed through operations. Defining these components contribute to the framework that implements all the basic concepts needed for integrating the control architecture.

The common components to most of the mechatronic systems are as following [3]:

- *I/O Generic Component*. This component provides input and output information of the controller, including the output of control instructions, checking information from mechatronic device, the input information, the relative feedback.
- *Control Law Component*. This component provides control to ensure absolute measurement, the control of a servo control system for compensation of errors. Furthermore, it also has the function of calculating the output from the controller and to set the cycle time.
- *Axis Control Component*. This component drives the mechatronic system on the given trajectory and speed from certain instructions, and information obtained from the I/O interface.
- *Human Machine Interface Component*. The Human Interface component provides a user with the ability to interact with not only the controller but also the machine or process operations. It is used to input system parameters, programme the machine and process operations, operate the machine or process being controlled, monitor the machine and process performance, display controller and process status, receive and display diagnostic information, etc.
- *Network Connection Component*. The Network Connection component ensures that information and data of the processes, machine and controller are transferred to the plant manufacturing information system when they are requested.

The above components can be treated as the main components in the mechatronic reusable generic database because their properties of extensibility, interoperability, portability and scalability.

4 Open System and Extended Generic Framework

Integration of various generic components forms an application. The generic components use the resources provided by other integrated and imported components to accomplish the task. The attributes and the parameters for these components are hidden. The implementation uses the predefined definition to obtain a target code. It looks simple, but the format of defined components is wide and general enough to capture the funtionalities required by wide range of mechatronic devices. The underlying principle and objective is to satisfy the requirements of an open system. The basic characteristics needed in an open system are maintainability, portability, understandability, and reusability [4]. Most of the changes in requirement are changes in the functionality, these changes can be disastrous to procedure based design. Changes in functional

requirements are easily accommodated in an object-oriented design by adding or changing operations in encapsulated objects. It is very well provided in this generic based system as the procedure of definition is completely separated from the procedure of implementation. Therefore, generic components are maintainable. Also, because the procedure mentioned above namely, definition and implementation are separated, the portability is accomplished. As the components are general interfaces they can be adapted to various hardware platforms and operating systems. Understandability states that the use of objects and messages allows an easily understandable to be developed and designed [5]. Manufacturing engineers and operators think of their systems in terms of machines, conveyors etc. in other words think their systems in terms of objects which compose them. The counterparts of objects are the generic components in object oriented environment. This makes it understandable to manufacturing engineers. As any component can be plugged into another application that needs its functionality without knowing its implementation details, reusability is guaranteed.

4.1 An extended generic framework for generic components

Since in the approach, section 2, it has been clear that to define a generic framework which enhances the integration of control so the focus is on two key aspects:

1. The mechanism for managing component processing (i.e. the services they offer / or demand) in a distributed environment.
2. The dynamic evolution of the framework adding new funtionalities or removing the existing ones.

It is to be noted that observing keenly, the generic components fall in two managerial compartments, namely service dependent components and the distributed communication dependent components. See Fig.2.

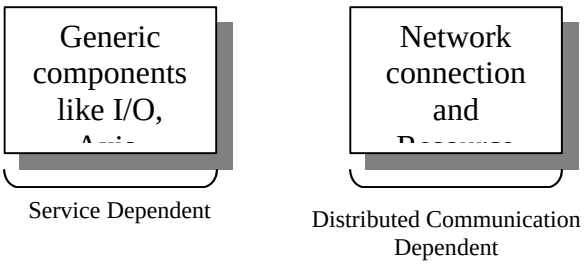

Fig.2. Compartments for the generic components.

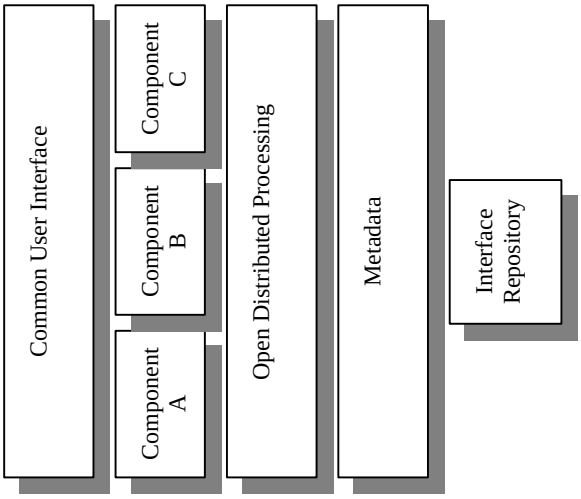

Fig.3. Open distributed integration of control.

Service dependent components include service interface definition, organisation, storage, service calls-based on communication, service execution, and services combination. Such components must be thought of and developed in order to respond to mechatronic application needs. In this perspective, a new service dependent component integration mechanism is introduced. It refines the way for component integration and communication and hence the communication dependent components.

Dynamic evolution of a framework to evolve and integrate with the components is resolved by the open distributing control integration. The new generic components publish themselves in metadata, which is an interface repository. This accomplishes the subscription of components to the newly evolved methods and operations available. Fig.3. clears the concept of open distributed integration of components.

In the following section, the design architecture of the generic components is described, emphasising the service component compartment as a means for realising and distributing the integration of control for the mechatronic device as well as detailing the interface repository for dynamic evolution of operations available on generic components.

5 Architecture for Integration of Generic Components

At the most basic level, CORBA (Common Object Request Broker Architecture) is a standard for distributed generic components. CORBA allows an application to request an operation to be performed by a generic component and for the results of the operation to be returned back to the application making the request. The application communicates

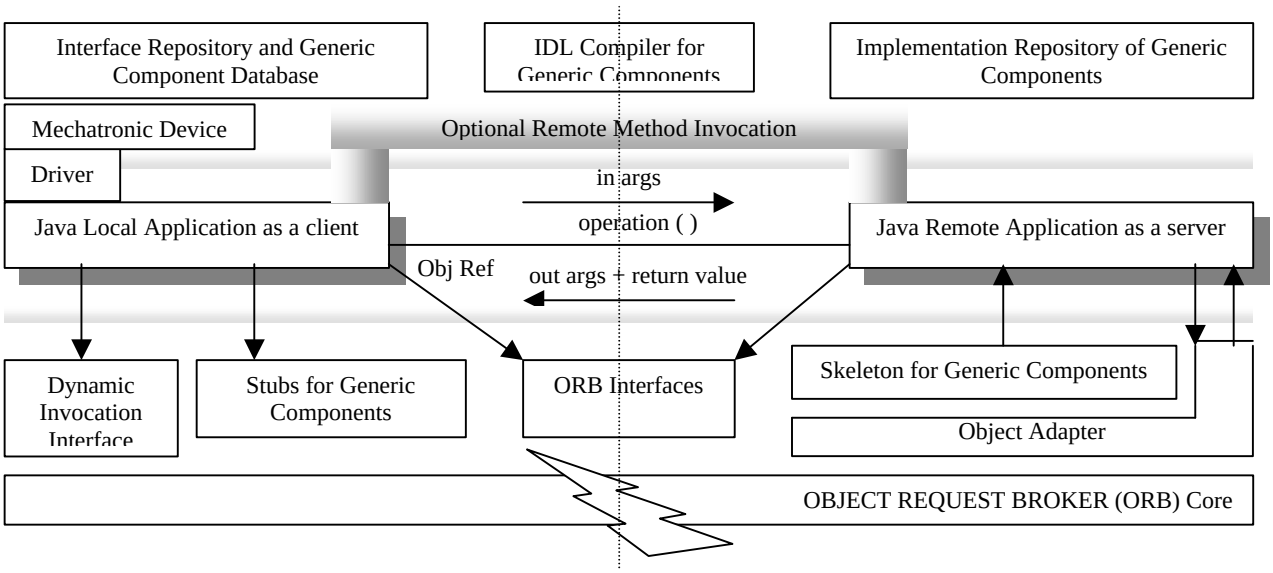

Fig.4. Architecture for integration of generic components based on ORB as core and component passing by reference.

with the distributed component that is actually performing the operation. This is basic client/server functionality, where a client issues a request to a server and the server responds back to the client. Data can pass from the client to the server and is associated with a particular operation on a particular component. Data is then returned back to the client in the form of a response.

- CORBA supports many existing languages and also the mixing of languages.
- Supports distributed standards as well as object-oriented standards.
- CORBA provides a high degree of interoperability. This insures that distributed components built on top of different CORBA products can communicate.

Java enables compatibility to most platform and also portability standards.

The architecture has been designed to guarantee the correct interchange of information between different implementations of the generic component cluster as in section 3.2. To do that, it defines a set of minima that must be met by all components, custom as well as commercial-off-the-shelf, that want to claim the conformance. This mandatory part of the architecture uses CORBA see Fig.4., as the backbone to provide the interchange of information regardless of the language used to implement the generic components.

The architecture also allows the use of other ways of communication, if available. For example, Java RMI (Remote Method Invocation) could be used when communicating from Java to Java, thus avoiding the conversion from Java components to CORBA components in the server, and from CORBA components back to Java components in the client, as clearly defined by marshalling procedures. To accomplish that, the architecture must provide additional standard, but optional, interfaces to cover these cases.

A series of libraries are defined to help engineers create implementations independent of CORBA, thus making designs less sensitive. A generic component database would help programmers who were not familiar with CORBA programming because they would use the components for the preferred language and still be able to communicate with other implementations in other languages.

The intercommunication is guaranteed because all the applications can be connected to any driver following the path through interface definition language and CORBA. However, an RMI optional path also is defined and allows Java remote applications to communicate directly with Java implementations that have this optional interface enabled.

5.1 The GCs interface description.

To guarantee the independence of the implementation from the description, there must be platform and language-independent interface to allow applications to interact regardless of how the architecture works underneath.

CORBA as the core architecture, and interface definition language; the way to express interface because of its characteristics. The interface definition language is the basis for all

implementations, and it specifies the minimum set of requirement that all the standard-complaint products must satisfy as mentioned in section 5.

With the interface definition language, it is possible to ensure the desired cross-platform compatibility without forcing programmers to learn a new language, and they can also use the generic components, defined, to help engineers with the architecture without requiring them to learn it. In the interface, the following must be specified clearly:

- The object-oriented model for every usable language.
- The interface to connect to remote servers, using interface definition language or libraries provided for different languages specified.
- The expected behaviour of remote servers. This part is language independent.

Servers can have different abilities depending on implementation. The level of the server to which an application is connected could be queried at runtime, allowing programmers to modify the behaviour of the application depending on that parameter.

Further, the interface allows dividing the system into two separate visual frames: the implementation frame and the service frame.

The Implementation Frame. When engineers create a design, they can model the application using the definition of the generic components, and they can rely on the behaviour of these components as it is specified. When the implementation time comes, they can use their language choice and then use the interface definition language to interact with the sources of information, or they can use generic components directly, if available for that language, so they do not need to be concerned about interface definition language.

From this frame, the developer does not know and really does not need to know how the model is implemented behind the components. The application is completely detached from the description.

The design and development of the application can focus on the application logic itself because the generic components ensure behaviour and hide it from implementation which accomplishes our goal of openness in portability as mentioned in section 4.

The Service Frame. From this frame, the engineers in charge of implementation can choose the appropriate generic component without having to worry about whether the application will use this component. The situation allows them to improve their products constantly, without fear of causing incompatibility problems with the applications, which again accomplishes the goal of extensibility, reusability and upgradeability as mentioned in section 4.

The only concern for the engineers is that their products must comply with the standard specification. The generic components define the expected behaviour and allow engineers to concentrate on the implementations.

5.2 Publishing about and subscribing to operations on generic components.

Components are stored in a specific format in the namespace and the client locates them by subscribing to component's object reference and then invoking an operation on this component available as a server. Fig.5. illustrates this operation.

1. A server of the generic component invokes method *bind* to associate a logical name with an object reference.
2. The name server adds this obj_ref/name binding to its namespace database.
3. A client to the generic component and hence the application invokes *resolve* to obtain an object reference with this name, and
4. The client uses the object reference to invoke methods or operation on the target generic component.

So the name server services both clients and servers. Servers export name/object bindings to the name server; clients then find these objects.

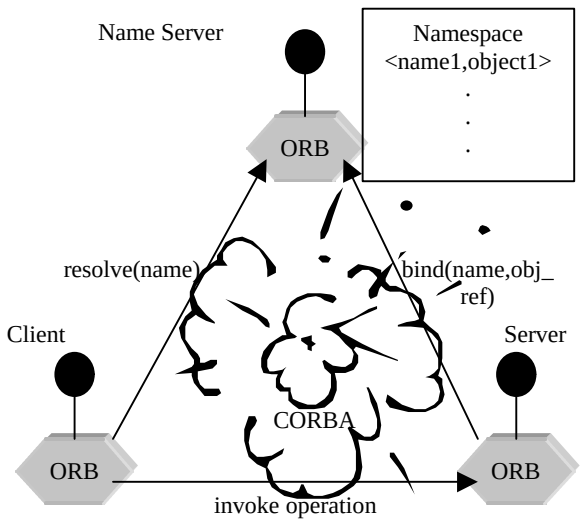

Fig.5. Interaction for publishing and subscribing.

6 Conclusions and Future Work

In this paper, the architecture for the generic components has been described. This also allows the components to be integrated and accomplish a mechatronic task in a distributed manner. The main feature is that it decouples the program designed for the application from fixed software and hardware of a mechatronic system. To achieve this, the concepts inherit important features of high level programming (modularity, encapsulation etc.) and also from publishing and subscription paradigm. Some of the generic components have been developed after modelling the mechatronic domain. Implementation of this system is in process and is being tested.

The experience acquired while designing the architecture and developing the components allow some preliminary conclusions:

1. The ease of development of generic components after the domain has been modularised has been demonstrated. To make them standard, means that to add parameters and attributes for most of the mechatronic devices have to be regularly updated. The software engineers with no previous experience with this architecture are able to successfully work on the implementation.
2. The communication between the generic components has been tested and compared to other tools.
3. Also, the language of implementation, Java enables compatibility to most platform and also portability standards.

Future work is going to include:

1. Extension of the generic components on the WWW (World Wide Web).
2. Visual tools for designing and programming.
3. Implementation of generic components on remote mechatronic system.
4. Extending the capability of core architecture for real time solutions.

References:

[1] N. Andersson, Design Principles for Open and Reusable Shop-Floor Control Software, *Computers in Industry*, Vol. 33, No. 2-3, 1997, pp. 285-293.

[2] I. Thomas and B. Nejmeh, Definition of Tool Integration for Environments, *IEEE Software*, Vol. 9, No.2, 1992, pp. 29-35.

[3] M. Singh, A. White, R. Gill and S. Prior, Development of IMS Tools for Mechatronic Systems, *Seventeenth National Conference on Manufacturing Research*, 2001, Awaiting for publication.

[4] C.M. Davidson and J. McWhinnie, The use of Object Oriented Methods in PLC Control System Software Development, *Proceeding of the Twelfth International Conference on CAD/CAM Robotics and Factories of the Future*, (R. Gill and C.S. Syan (ed)), 1996, pp. 212-218. Middlesex University Press, London.

[5] S. Adiga and P. Cogez, Towards an Object-Oriented Architecture for CIM Systems, *Object-oriented Software for Manufacturing Systems*, (S. Adiga (Ed)), 1993, Chapman & Hall.