Copyright © 2002 IFAC
15th Triennial World Congress, Barcelona, Spain

TOWARDS A CONTROL SOFTWARE DESIGN ENVIRONMENT USING A META-MODELLING TECHNIQUE

D.N. Ramos-Hernandez, I. Zubizarreta, P.J. Fleming, S. Bennett and J.M. Bass§

Department of Automatic Control and Systems Engineering, The University of Sheffield, Mappin Street, Sheffield S1 3JD, UK.
E-mails: d.n.ramos-hernandez@sheffield.ac.uk, COP00IZ@sheffield.ac.uk, P.Fleming@sheffield.ac.uk and S.Bennett@sheffield.ac.uk.
§*School of Informatics, University of Wales, Dean Street, Bangor, Gwynedd. LL57 1UT, UK.*
E-mail: Julian.Bass@chordiant.com

Abstract: The novelty of this paper is mainly the integration of multi-disciplinary software tools into a control software design environment, namely the Integrated Design Notation (IDN). The IDN supports the design, development and implementation of decentralised distributed control systems. This new environment is based on the UML meta-model standard. The translation process to integrate a control software tool (Simulink) and a process control standard (SFC) into the IDN are described. An approach for generating Java code automatically from UML also is proposed. The Java code generated is tested on a real-time target hardware architecture. *Copyright © 2002 IFAC*

Keywords*: Distributed control, control systems, object-oriented technology, meta-modelling, process control, real-time systems.*

1. INTRODUCTION

This paper describes a new development of a control software design environment, namely the Integrated Design Notation (IDN). This is part of the "Process Control System Integration" (PiCSI) programme funded by the UK EPSRC Systems Integration Initiative and involves significant academic and industrial collaborators. The IDN is a major advance on the Development Framework previously reported at IFAC conferences (e.g. Browne, *et al.*, 1997; Hajji, *et al.*, 1996). This new version is based on object-oriented (OO) technology and uses a meta-modelling approach.

At present, the development process to achieve decentralised distributed control systems for the process and manufacturing industries is very complex. The basic technology used to implement control systems is software technology: however, exploiting software flexibility increases complexity which increases the probability of failure (Sanz and Alonso, 2001). This is exacerbated by a lack of software tools to support all development lifecycle phases: simulation/modelling, development and implementation. Software packages that exist generally do not conform to standards and this makes integration difficult. Integrating different tools in one environment should result in faster development cycles (since code can be reused from one tool to another), lower integration costs, improved productivity (with the potential for a single information system) and reduced maintenance costs.

The IDN is based on object-oriented technology since control systems components map naturally onto the concept of objects. Sanz and Alonso (Sanz and Alonso, 2001) described how OO technology is becoming the technology of choice to build complex real-time systems because it provides better mechanisms for handling complexity. In addition, OO technology facilitates flexibility, adaptability and reusability of the developed software objects.

At present, similar work to the project PiCSI has not been found. The most closely related work is by Dias, *et al.* (2001). Here, the authors present an approach called MOSYS, a methodology for developing distributed real-time applications based on distributed autonomous objects. Their approach basically provides a method for automatic identification of possible design objects architectures and includes testability support during the design phase. However, this approach does not support the

integration and co-simulation of different tools, and its design is focused in choosing an appropriate architecture. Their approach also offers support for automatic code generation for a real-time operating system (target language is AO/C++). Meanwhile PiCSI offers automatic Java code generation, 'write once, run anywhere' supporting hard real-time systems.

Currently, an initial field trial for the PiCSI project prior to the final implementation is being carried out with the aim of providing an early assessment of the research. The target for this field trial is a servomechanism on which will be tested several controller structures (e.g. P or P+I). The translation processes of Simulink and Sequential Function Chart (SFC) to IDN (UML) and IDN to automatically generate Java code will be applied to these controller structures. The code generated will be then executed and evaluated in the servomechanism.

The organization of the paper is as follows. The design of the Integrated Design Notation (IDN) is presented in Section 2. The servomechanism and real-time target hardware architecture are described briefly in Section 3. Section 4 describes the translations to integrate Simulink and SFC into the IDN and the automatic Java code generation approach. Section 5 concludes the paper.

2. THE INTEGRATED DESIGN NOTATION (IDN)

The aims of the Integrated Design Notation (IDN) are to establish and redefine open infrastructures that enable integration and co-simulation of continuous and state-event system models (Bass, 1998a; Bass, 1998b; Turnbull, 2000).

The IDN is based on the Unified Modelling Language (UML), which is the Object Management Group (OMG) standard for modelling object-oriented systems. This modelling language has a rich set of notations and semantics, which can be applicable to a wide set of modelling applications and domains, for example real-time embedded systems (Douglass, 1998). Although UML is a well-defined and flexible modelling language it lacks certain modelling techniques for real-time systems and embedded systems. Currently, the OMG has started a new initiative to recommend UML Real-Time extensions, the Real-Time Analysis and Design Group (RTAD) (Moore, 2001).

Using a graphical representation of the design enables the relationships between components of a system to be better understood. Thus, with modelling languages it is possible to provide a description of the implementation of the system and also to model the architecture of the system independent of the implementation language (Harrison, *et al.*, 2000).

UML is being used in three ways in this work:

1. The "4+1" software architecture and the many models and diagramming techniques (Quatrani, 1998) in UML can be used to represent models from the control domain. For example, a Simulink model can be captured in a class diagram.
2. UML is also being explored as a meta-modelling facility for the Integrated Design Notation (IDN). A meta-model such as UML defines a language to write models at level model.
3. Another aim of the work is to generate code (Real-time Java) from 'pictures' (Simulink, Stateflow and IEC 1131-3). This speed up the development process of control models used in a manufacturing plant or control system. In addition, the standardisation of UML and the availability of CASE tools facilitates this work and is expanded in section 4.

The IDN consists of three main models: the requirements model, the software task model and the target hardware model:

- The *requirements model* provides policies and mechanisms for the hierarchical integration of the selected domain-specific views, such as transfer-function block diagrams (Simulink), state charts (Stateflow) and IEC 1131-3 standard process control notations. Translation rules for converting each view into the requirements model are defined.
- The *target hardware model* enables systems specified in the requirements model to be manipulated into a form that may then be implemented. In this model, the target hardware elements of the application are modelled.
- The *software task model* integrates information obtained from the requirements and target hardware models in a form which facilitates automatic generation of Java source-code. This model enables temporal analysis of the system under development, prior to implementation.

Currently, there are several Computer-Aided Software Engineering (CASE) tools supporting UML. In general, a CASE tool might cover strategic planning, through domain analysis, system analysis, design, implementation (code generation), and testing, from an object-oriented perspective (Coad and Yourdon, 1991).

Since the IDN environment must facilitate the incorporation of new software packages in the future as well as accessing legacy software, Java on its own or combined with a distributed object technology, such as CORBA (Common Object Request Broker Architecture), allows ready integration. The CASE tool chosen to support the IDN is Rational Rose Enterprise 2000 (Rational Software). Unfortunately, real-time extensions are not supported in this version.

The Rational Rose RT version is aimed at the telecom sector and does not have the Rational Rose Extensibility Interface (REI), which is the main advantage of Rose Enterprise in term of research. The REI is the common set of interfaces used by Rational Rose Script and Rational Rose Automation to access Rational Rose (Rational Rose, 2000). Using Rose scripting language it is feasible to automate and increase the functionality of Rose. For example, Rose script language can be used to translate different tools to UML notations or this language can be used to create an automatic Java code generation.

abstraction. It works well for real-time system design (Moore, 2001). This diagram was chosen to define the continuous system view (Simulink and FBD language of the IEC 1131-3) of the Requirements Model because this shows the structure of the system in terms of classes and objects, and how they relate to each other. This structure is very similar, for example, to a Simulink diagram.

The *Statechart diagram* describes the behavior of a class. This diagram shows all possible states that objects of the class can have and which events cause the state to change. The statechart diagram was

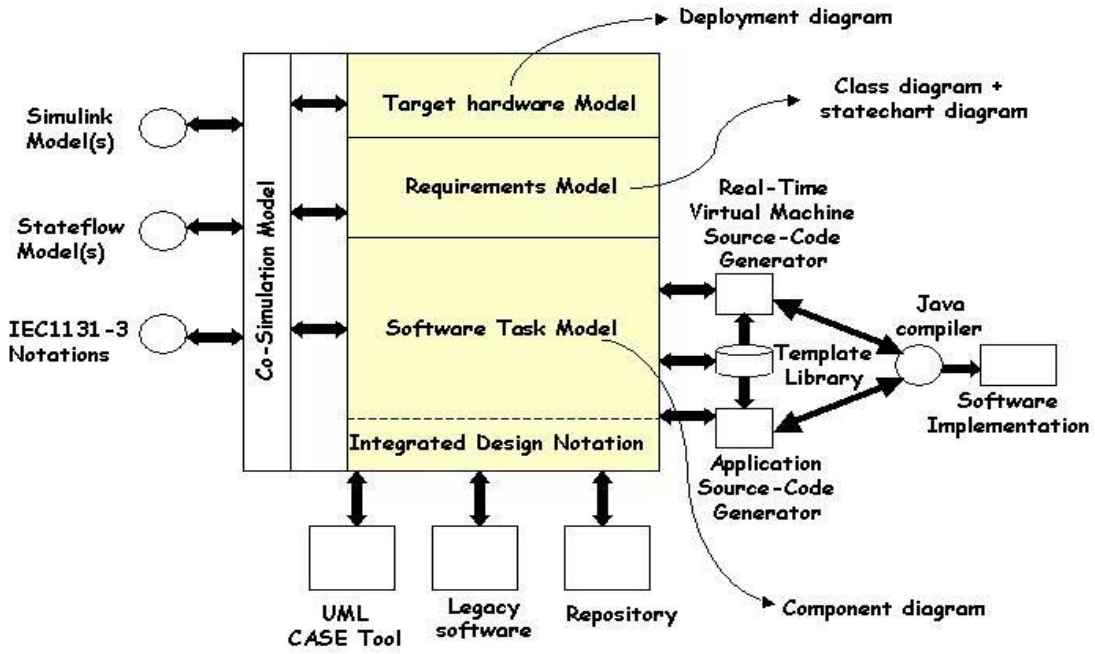

Fig. 1. Integrated Design Notation.

Fig. 1 shows the design for the IDN environment. In this, the three models of the IDN (target hardware model, requirements model and software task model) are presented. Through the IDN the different tools can access different views of the system directly or indirectly. Also, the IDN allows the replacement or integration of tools. The UML CASE tool (Rational Rose) can be accessed via the IDN as well as legacy software and a repository tool. Finally, a Java source code generator application and real-time virtual machine tools are illustrated which are connected to the template library and to a Java compiler producing executable code for a target distributed architecture allowing reverse engineering of the system at a later stage.

To describe the three main models of the IDN, Fig. 1 shows the UML diagrams that were chosen. The *class diagram* is concerned with static structure and the design of good partitioning (encapsulation) and

selected to define the state-event system view (Stateflow and SFC language of the IEC 1131-3) of the Requirements Model.

The elements of a *Deployment diagram* match hardware elements such as nodes or processors, however for a complex architecture this diagram is inadequate. Thus, the Target hardware Model will be extended based on the Deployment diagram.

A *Component diagram* shows the dependencies between software components in the system (Bennett, *et al*., 1999). This diagram was chosen because it contains elements (e.g. components) required to define the Software Task Model. However, the Software Task Model will combine information between the other two IDN models.

In this paper, an early integration of the Simulink tool and the SFC into the IDN is described. The

completed environment is planned for September 2002.

3. A CONTROL SYSTEM APPLICATION - DC SERVO

A servomechanism (DC Servo) is used as the application for the initial testing of the environment's capabilities. The servo provides position and speed control using typical industrial techniques. It is also used as a flexible tool for the practical design, operation and application of controllers. Thus, this servomechanism allows the testing of alternative controller structures such as P, PI or PID, which can be applied externally using dedicated control applications such as Simulink (MATLAB). In this paper, a *proportional controller* structure which controls the servomechanism is used to illustrate the translation process of Simulink to UML and UML to Java in section 4. The start ON/OFF of the servomechanism was modelled using the SFC language and this was translated into UML. The DC Servo is connected to the host (laptop computer) via a one-wire network, where the Control Systems Centre at UMIST has specially developed an AD/DA board with one-wire technology.

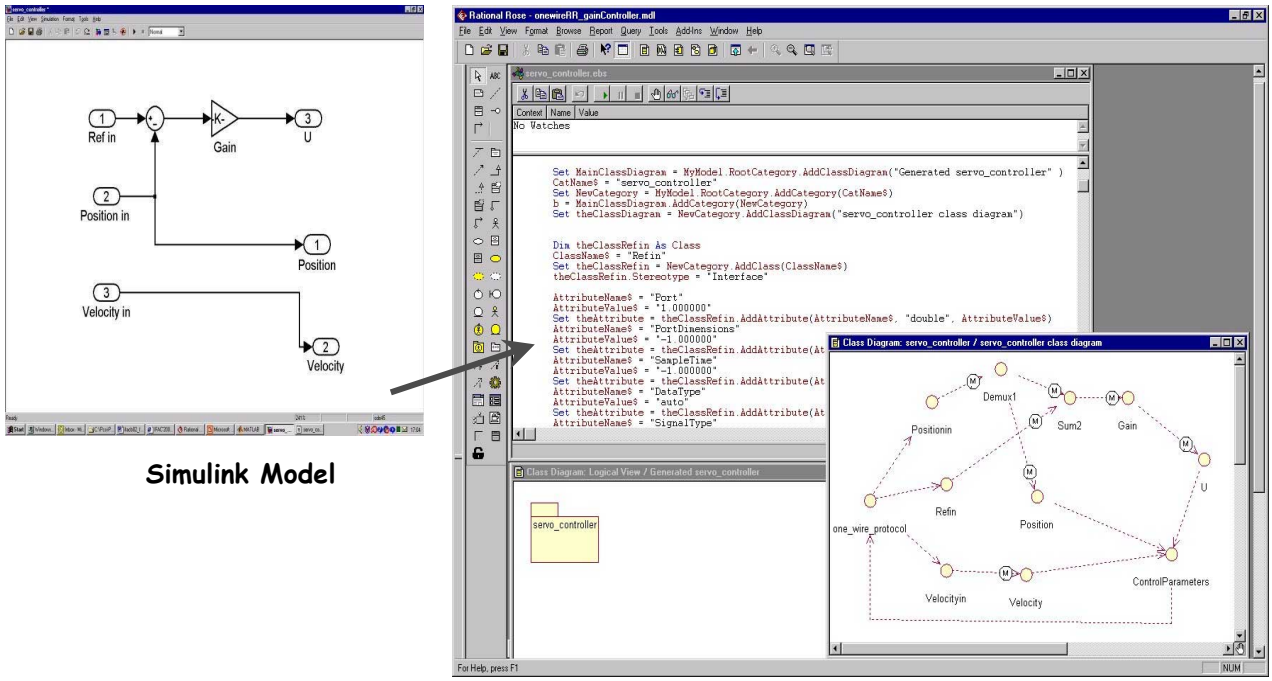

Fig. 2. Class Diagrams of the proportional controller.

4. TRANSLATIONS

The translations of the different tools into the IDN are based on the Rational Rose Extensibility Interface (REI). At present, a complete generation of Java code from pictures has been done with only the Simulink tool.

4.1 Simulink to UML (sim2uml).

The control structure of a controller is generally complicated, however, using OO technology to map hardware components as objects simplifies the representation of the structure of the controller.

A C program was developed for this translation. The program has been tested with several controllers, it handles subsystems and several Simulink blocks (e.g. Inport, Output, Gain, Sum, Saturate, Demux, TransferFcn). This program extracts blocks and connections from the Simulink model and generates a script file. The script file is then executed in Rational Rose and translated into UML class diagrams. Fig. 2 shows the syntax of the script file and the class diagrams generated with this file for the proportional controller. In the UML diagrams the proportional controller is represented within a package and inside this package each block is represented by a class (small circles). Dashed arrows mean dependency or instantiation and represent connections between blocks.

4.2 SFC to UML (sfc2uml).

The Sequential Function Chart (SFC) is one of the five languages supported by the IEC 1131-3 notation (a tool that supports this notation is ISaGRAF PRO developed by Altersys.). This is a graphical language for depicting sequential behaviour of time and event-driven control system (Lewis, 1998). The translation of the SFC language is based on the Rational Extensibility Interface (REI) as well. A script file is

generated after running a Java program, which uses iterative loops to read information from an ISaGRAF PRO text file (*.stf). The text file contains the steps, actions and transitions of a system. Once the script file is generated, this is executed within Rose generating a Statechart (SC) diagram.

4.3 UML to Java.

Code generation tools of model representations are very useful to maintain consistency between a model and its implementation (Harrison, *et al*., 2000). The majority of the UML CASE tools support the generation of skeletal implementation code either directly or by exporting models in a standardized format, such as XMI (XML MetaData Interchange).

Harrison, *et al*. (2000) developed a UML-to-Java code generator tool. The tool they described converts UML class diagrams expressed in XMI into Java classes following a methodology. This methodology places fewer constraints on the design allowing the use of UML constructs such as multiple generalization and association classes. Peltier, *et al*. (2000) proposed a two-space framework to transform a model to another one with abstract transformation rules, which are expressed on a meta-model. From these transformation rules, they can generate a program that applies these rules to models. Thus, they used this solution to transform a UML model to a Java model.

The automatic code generation that translates UML diagrams into Java code is still under development. The main objective of this translation is to generate source code based on a Template Library, which contains the different blocks (classes) of a Simulink diagram, such as integrators, gains, multiplexes, etc. This Template Library (package) is then imported from other Java files which contain the connections between blocks. These files use the concepts of multithreading and pipes for communication between threads (Horstmann and Cornell, 2000). These concepts are the most reasonable routes for multi-processor systems. The approach considered to develop automatic Java code generation was to use the Rational Rose Java code generator and Javadoc. This approach involves first generating the Java code using the Rose Java tool. Then, it is necessary to create the actual functionality for the application. To produce this functionality such as multi-processing communication code and the import of the Template Library as well as its generation, Javadoc seems a straightforward approach. Using a few doclets, apart from obtaining classes, methods, fields and tags information, it is possible to create and extend this functionality. Fig. 3 shows this generation of code using Javadoc and Rational Rose.

Another option to automatically generate Java source code was evaluated, which consisted of modifying an existing scripting file. This option was relatively easy, however the script language is more limited than Java.

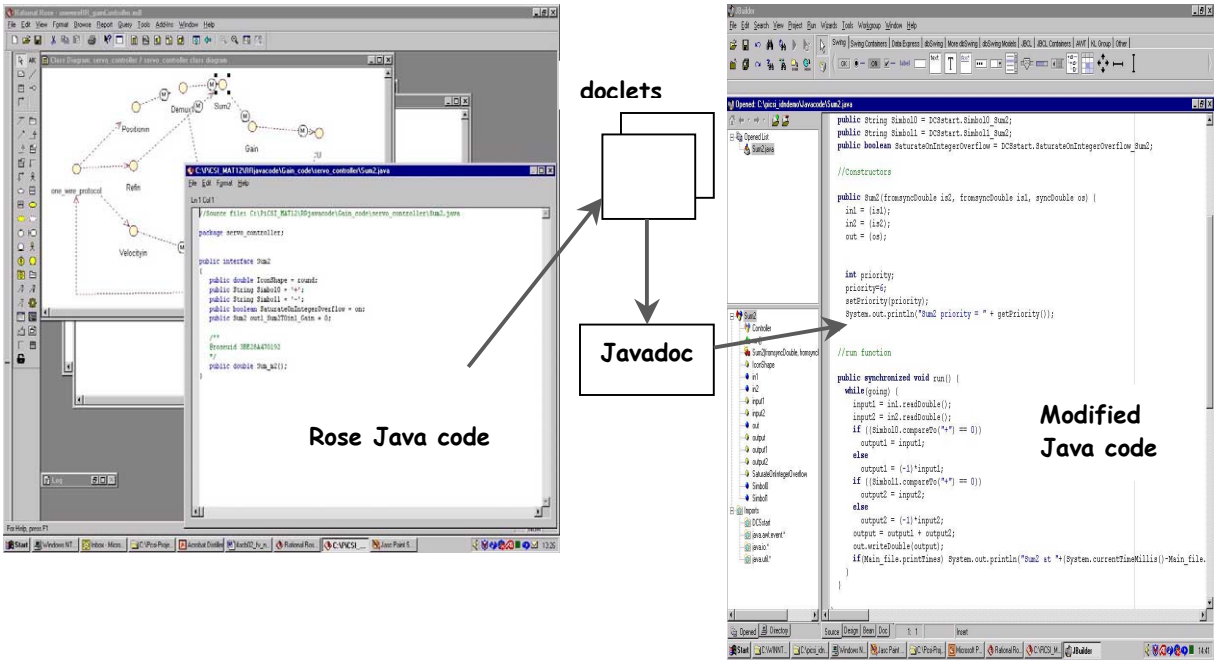

Fig. 3. Code generation using Javadoc and Rational Rose.

The manipulation of class descriptions of XMI is extremely similar to the Javadoc API. However, the XMI requires a toolkit API and Javadoc is distributed as part of the Java SDK (Pollack, 2000).

It was decided to use the Rational Rose Java code generator and Javadoc approach, which has shown potential to automatically generate the source code for a Simulink model. Currently, the Java source code for several controllers (proportional, lead, state

space and observer) for the servomechanism and a PID controller for the final application (extruder control) has been generated successfully using this approach.

5. CONCLUSIONS AND FUTURE WORK

A control software design environment, the IDN has been presented in this paper. This is targeted to improve the development and performance of decentralised distributed control systems for the process and manufacturing industries. In order to provide an early assessment for the research an application of a DC Servo has been described where an object-oriented approach simplifies the representation of the structure of the controller. The translation process of Simulink and SFC to UML, and the automatic Java code generation from UML have been described. The code generated is then executed and evaluated in the servomechanism. Further work will include enhance the *sfc2uml* translation tool to support a hierarchical structure and extending the Template Library for the *sim2uml*. The real-time target hardware and Java real-time requirements have been addressed by the University of Wales, Bangor and the controller structures by UMIST.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of UK EPSRC (Grant GR/M55299) and collaborators in industry, University of Wales, Bangor and UMIST.

REFERENCES

Bass, J.M. (1998a). Proposals Toward an Integrated Design Environment for Complex Embedded Systems, *Euromicro 6th Workshop on Parallel and Distributed Processing*, Madrid, Spain, January 1998, pp. 273-8.

Bass, J.M. (1998b). An Open Environment for the Specification, Design and Code Generation of Control Algorithms, *IEE Colloquium on Open Control in Process and Manufacturing Industries*, London, May 1998, pp. 7/1 - 7/4.

Bennett, S., S. McRobb and R. Farmer (1999). *Object-Oriented Systems Analysis and Design using UML*. McGraw-Hill Publishing Company, England.

Browne, A.R., J.M. Bass and P.J. Fleming (1997). A building-block approach to the temporal modelling of control software, *Proc 4th IFAC Workshop on Algorithms and Architectures for Real-Time Control AARTC 97*, Portugal, *pp* 433-438.

Coad, P. and E. Yourdon (1991). *Object-Oriented Design*. Prentice Hall, New Jersey, USA. Chapter 9.

Dias O.P., I.M. Teixeira, J.P. Teixeira, L.B. Becker and C.E. Pereira (2001). On identifying and evaluating object architectures for real-time applications, *Control Engineering Practice*, 9, pp. 403-409.

Douglass, B.P. (1998). *Real-Time UML Developing Efficient Objects for Embedded Systems*. Addison-Wesley. Reading, Massachusetts, USA.

Hajji, M.S., A.R. Browne, J.M. Bass, P. Schroder, P.R. Croll and P.J. Fleming (1996). A prototype development framework for hybrid control system design, *Proc 13th World Congress of IFAC*, Vol. O, pp 459-464.

Harrison, W., C. Barton and M. Raghavachari (2000). Mapping UML Designs to Java, *ACM SIGPLAN Notices 35*, No. 10, 178-187.

Horstmann, C.S. and G. Cornell (2000). *Core Java 2, Volume II – Advanced Features*. Sun Microsystems Press, A Prentice Hall Title.

Lewis, R.W. (1998). *Programming industrial control systems using IEC 1131-3*. Revised edition. IEE Control Engineering Series 50. The Institution of Electrical Engineers.

Moore, A. (2001). Real-Time UML, *Embedded System Engineering*, December/January, pp.48-49.

Peltier M., F. Ziserman and J. Bezivin, (2000). On levels of model transformation, http://www.gca.org/papers/xmleurope2000/papers/s36-02.html (Access date: 19/07/01).

Pollack, M. (2000). Code generation using Javadoc. Extending Javadoc by creating custom doclets. *JavaWorld*, August 2000. http://www.javaworld.com/javaworld/jw-08-2000/jw-0818-javadoc.html (Access date: 15/09/00).

Quatrani, T. (1998). *Visual Modeling with Rational Rose and UML*. The Addison-Wesley Object Technology Series. Grady Booch, Ivar Jacobson, and James Rumbaugh Series Editors.

Rational Rose (2000). *Rose Extensibility User's Guide*. Rational Software Corporation. Version 2000.02.10.

Sanz, R. and M. Alonso (2001). Corba for Control Systems. *Annual Reviews in Control 25*. (J.J. Gertler (ed)), pp 169-181.

Turnbull, G., (2000). Improving the bottom-line through open standards. *The Application of IEC 61131 in Industrial Control. Improve your Bottom-Line Through High Value Industrial Control Systems*. IEE Control Division, Birmingham, UK.