Monograph:
Ramos-Hernandez, D.N., Fleming, R.F. and Bennett, S. (2001) Toward an Object-Oriented Process Control Software Design Environment. Research Report. ACSE Research Report 795 . Department of Automatic Control and Systems Engineering

Department of
AUTOMATIC
CONTROL
and SYSTEMS
ENGINEERING
UNIVERSITY of SHEFFIELD

TOWARD AN OBJECT-ORIENTED PROCESS CONTROL SOFTWARE DESIGN ENVIRONMENT

D.N. Ramos-Hernandez, P.J. Fleming and S. Bennett

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, P.Fleming@sheffield.ac.uk, S.Bennett@sheffield.ac.uk.

Research Report No. 795

July 2001

Abstract:
The design of a control software design environment, namely the Integrated Design Notation (IDN) is presented. IDN supports the design, development and implementation of decentralised distributed control systems. A cable extrusion process is targeted as a demonstrator application, where object-orientation technology is expected to facilitate the improvement of extruder control in a distributed environment. The IDN is based on the Unified Modelling Language (UML). A CASE tool supporting UML is integrated with the IDN. The translation to integrate a control software tool (Simulink) and options to generate automatic Java code are described.

Keywords: *Distributed control, object-oriented technology, 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 EPSRC. The IDN is an advanced version of the Development Framework previously reported at IFAC conferences (e.g. Browne *et al.*, 1997, Hajji *et al.*, 1996) as this new version is based on object-oriented (OO) technology.

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.

The organization of the paper is as follows. The design of the Integrated Design Notation (IDN) is presented in Section 2. The target application, a cable extrusion process, where the control can be improved and developed with the object-oriented environment is described in Section 3. Section 4 describes the translation to integrate Simulink into the IDN and the automatic Java code generation options. 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, 1998).

The IDN is based on the Unified Modelling Language (UML), which is the 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).

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.
2. UML is also being explored as a meta-modelling facility for the Integrated Design Notation (IDN)
3. Another aim of the work is to generate code (Real-time Java) from 'pictures' (Simulink, Stateflow and IEC 1131-3) which describe the control model of a manufacturing plant. The

standardisation of UML and the availability of CASE tools facilitates this work and is expanded in section 4.

Therefore, the Unified Modelling Language (UML) and the IDN are being used to support the design, development and implementation of decentralised distributed control systems.

Simulink is a software package for modelling, simulating and analysing dynamical systems (SIMULINK, 1999). Stateflow is a graphical design and development tool for complex control and supervisory logic problems (STATEFLOW, 1999). IEC 1131-3 is a standard that defines how control systems such as programmable logic controllers (PLCs) could be programmed. This allows industrial instrumentation and control systems engineers to build large systems using equipment from different manufacturers (Lewis, 1998).

The IDN consists of three models: the requirements model, the software task model and the architectural 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 *architectural model* enables that 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 architectural 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. Thus the features considered to select a CASE tool to support the IDN are the ability to reverse-engineer in Java, CORBA/IDL (Interface Definition Language) support, repository support and finally HTML documentation. A software product that supports all of these features is Rational Rose Enterprise 2000 (Rational Software). Unfortunately, real-time requirements are not supported in this version. The main advantage of Rose is its Rational Rose Extensibility Interface (REI). 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 considering real-time requirements.

Figure 1 shows the design for the IDN environment. In this, the three models of the IDN (requirements model, software task model and architecture model) are presented. Through the IDN the different tools can access the different information 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 in a later stage.

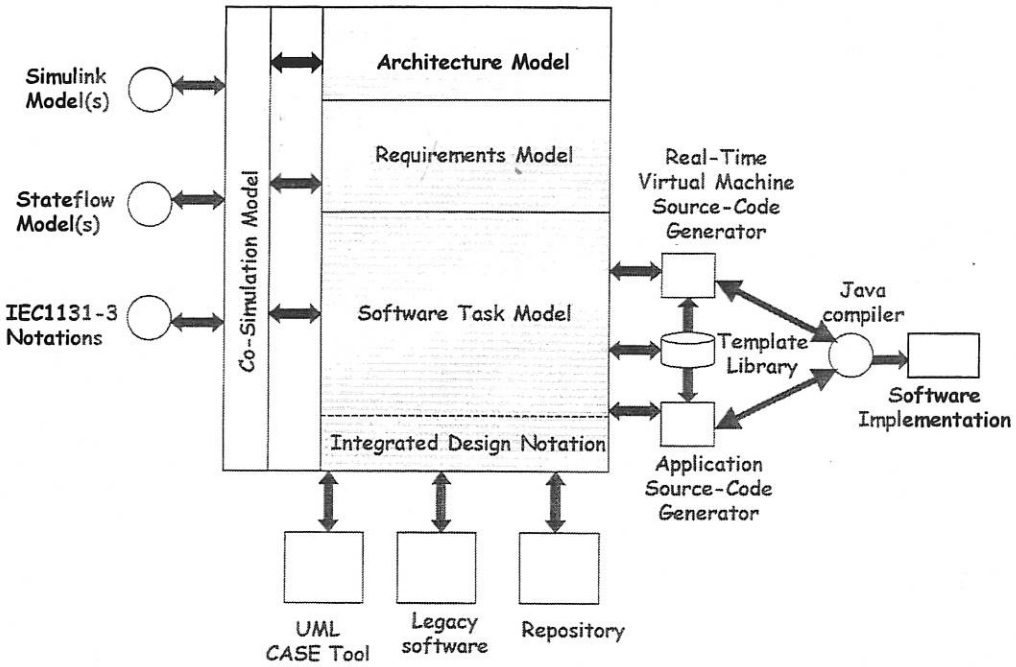

Fig. 1. Integrated Design Notation.

To describe the three models of the IDN, the Table 1 shows the UML diagrams that were chosen:

Table 1. UML diagrams chosen for the IDN Models.

IDN Model	UML diagram
Requirements Model	Class diagram
Architecture Model	Deployment diagram
Software task Model	Component diagram

A Class diagram provides both a high-level basis for systems architecture, and a low-level basis for the allocation of data and behaviour to individual classes and object instances, and ultimately for the design of the program code that implements the system. This diagram was chosen to define the Requirements Model because this shows the structure of the system in terms of classes and objects, and how they related each other. This structure is very similar for example to a Simulink diagram.

A Deployment diagram shows the configuration of run-time processing elements and the software components and processes that are located on them. The elements of the Deployment diagram match the target hardware elements (e.g. nodes or processors) of the Architecture Model.

A Component diagram shows the dependencies between software components in the system (Bennett *et al.*, 1999). This diagram was chosen because contains elements (e.g. components) required to define the Software Task Model.

In this paper, an early integration of the Simulink tool into the IDN is described. The completed environment is planned for 2002.

3. A PROCESS CONTROL APPLICATION - CABLE EXTRUSION PROCESS

The target application for testing the environment's capabilities is a cable extrusion process. This process is simply the forcing of thermoplastic (melted from pellets) through a restricted opening to form a continuous-length shape. The extrusion is usually cut to length on-line and then notched and drilled for additional openings. Structural details and surface treatments are limited to the forming direction.

The plastic material is basically fed through the transport section onto a rotating screw via the hopper or feeder. The extruder which converts the plastic granules into the homogenous melt is quite similar to the one used in injection moulding. The plastic is heated slowly as it is moved and pressed forward towards the die. The melt is forced and compressed through the die. Once cooled with either water or other coolants, the extrusion is sized and cut to desired length[1]. (See Figure 2)

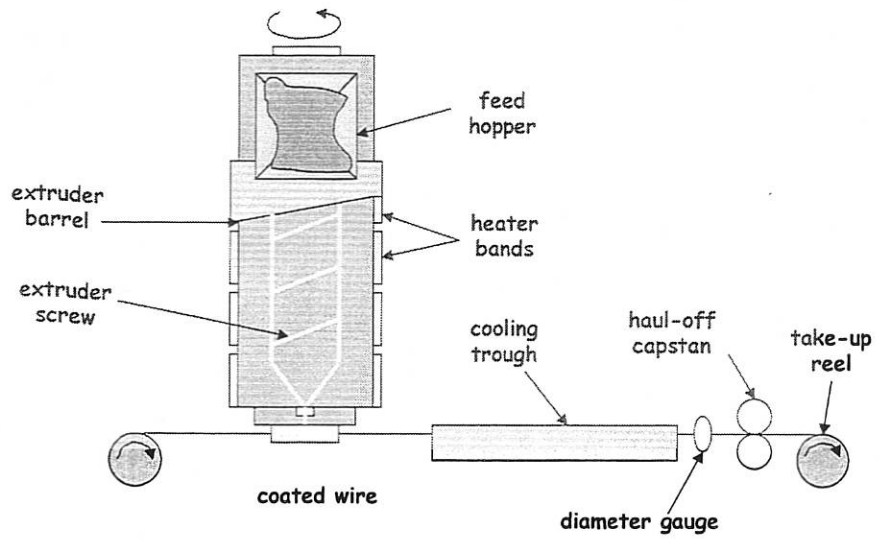

Fig. 2. Cable extrusion process.

[1] http://www.core77.com/resource/plastique/extrusion.html

The main aims of extruder control improvement are to minimise start-up scrap and maximise production line speed while maintaining quality. Important issues to observe in extruder control are the effect of interaction between extruder zones, controlling the speed of the screw and the heat generated by the screw.

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

The extruder control will be modelled using the different views (Simulink, Stateflow and IEC1131-3) in a complementary form. Additional models of the application will be obtained using the IDN, such as the software task model (which enables temporal analysis of the controller) and the architectural model (which contains the target architecture of the application). A combined simulation of the different views and models will be obtained via the Co-Simulation model. Once the extruder is modelled and simulated, it is planned to produce Java executable code for the specific distributed architecture. The distributed architecture is shown is Fig. 3. This consists of three nodes connected to the host via Ethernet. The nodes, PC104s and PC3000s are connected via Modbus in an early implementation and via Probifus afterwards. PC104 is an industry standard (IEEE-P996.1) for compact embedded PC-compatible modules used within manufacturing, machinery and industrial process or plant control applications. PC3000 is a programmable controller; this is configured using an IEC1131-3 PC based tool. Both Modbus and Profibus are fieldbus standards as well as Ethernet.

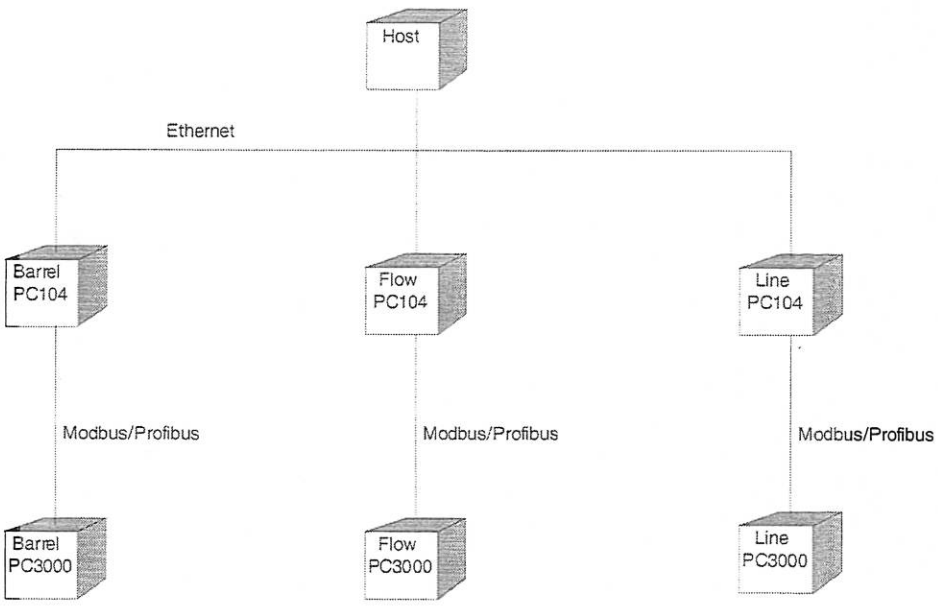

Fig. 3 Distributed architecture.

4. TRANSLATIONS

The translations are based on the Rational Rose Extensibility Interface (REI). Figure 4 shows two translations, the first one to integrate Simulink into the IDN and the second one to generate Java code automatically from UML. The translations shown in this paper are from a PID subsystem of the extrusion process control.

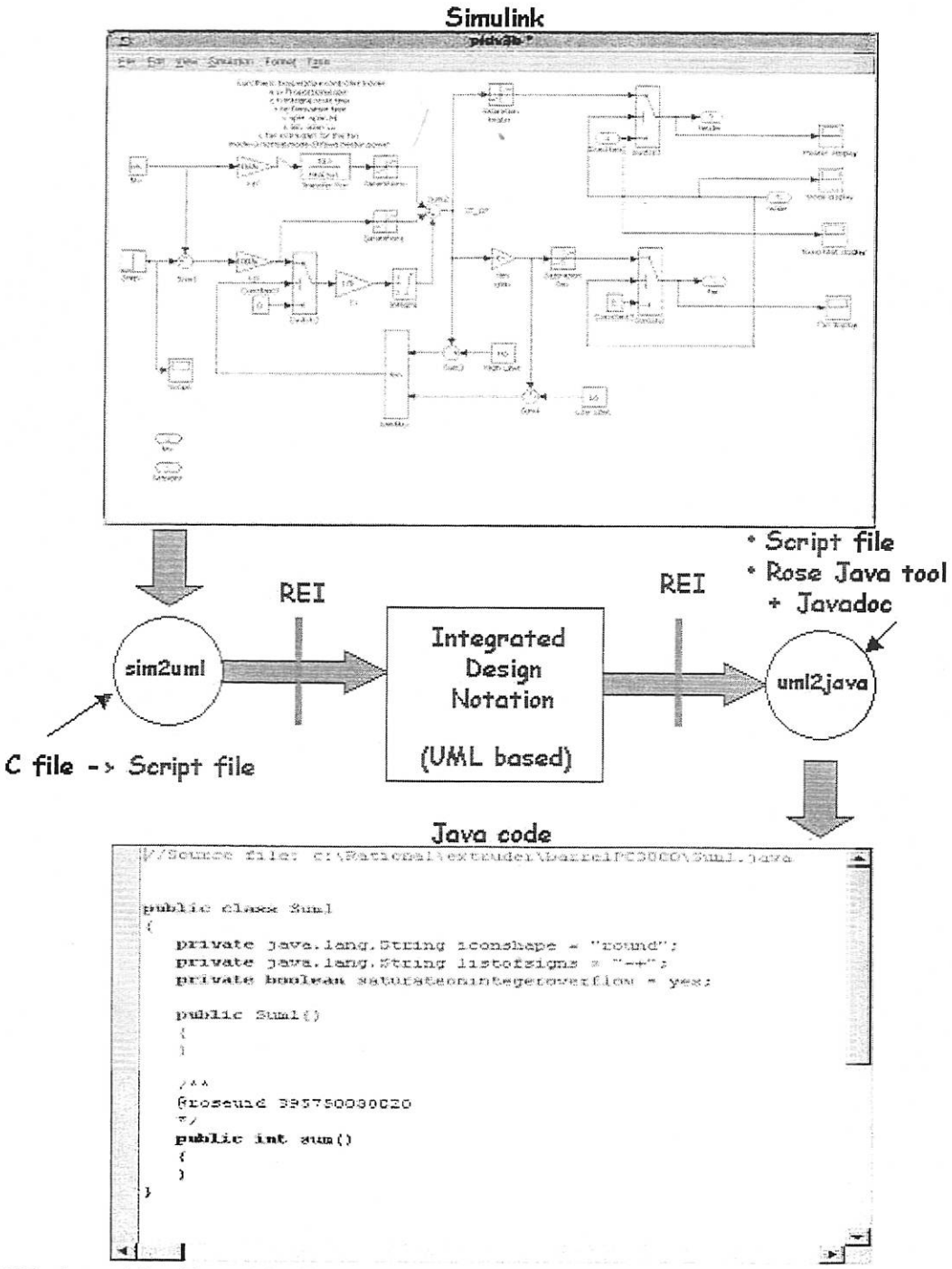

Fig. 4. Translations to integrate Simulink tool into the IDN and to generate Java code.

4.1. Simulink to UML

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. 5 shows the syntax of the script file and the class diagrams generated with this file. In the UML diagrams the PID subsystem is represented within a package and inside this package each block is represented by a class. Inport and Outport Simulink blocks are represented by interfaces (small circles). Dashed arrows mean dependency or instantiation and represent connections between blocks.

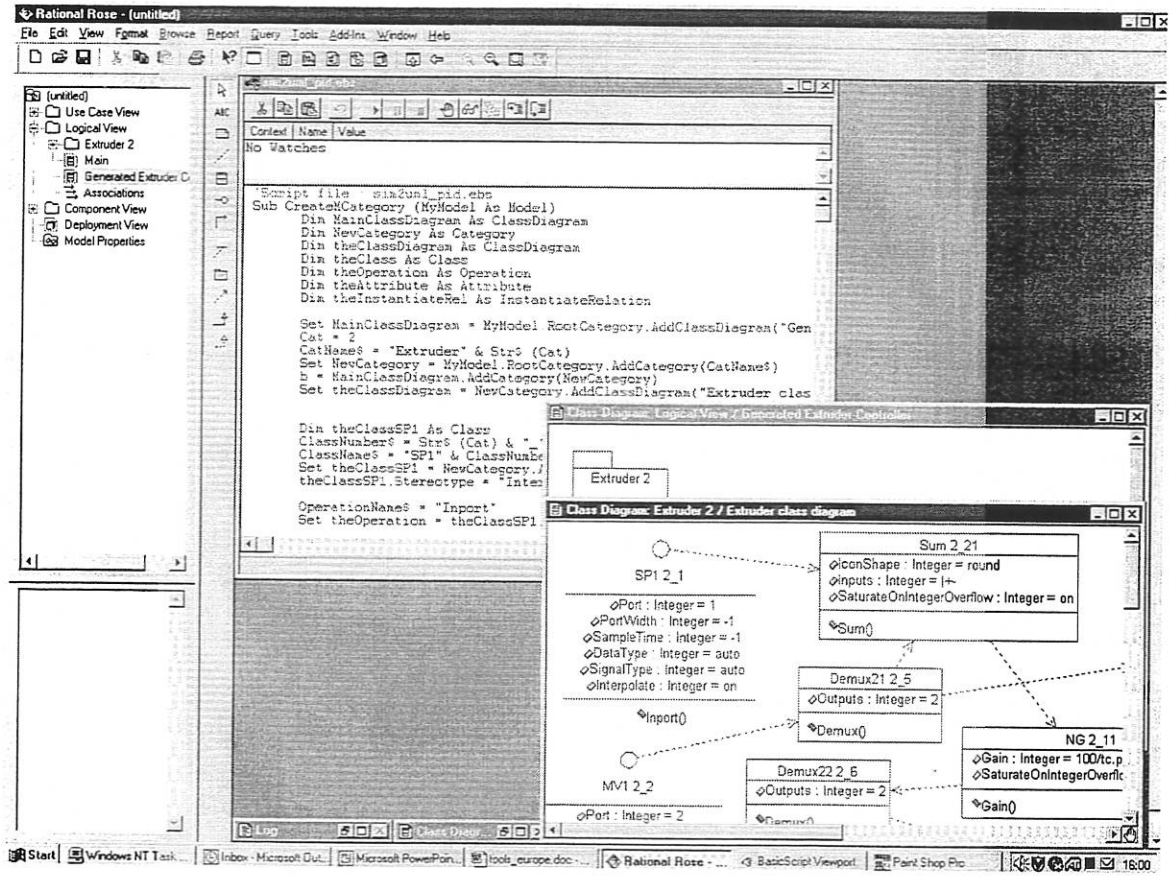

Fig. 5. Translation from Simulink to UML: script file and class diagrams.

4.2. UML to Java

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 containing 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. Fig. 6 shows an integrator Simulink block translated into Java within the Template library.

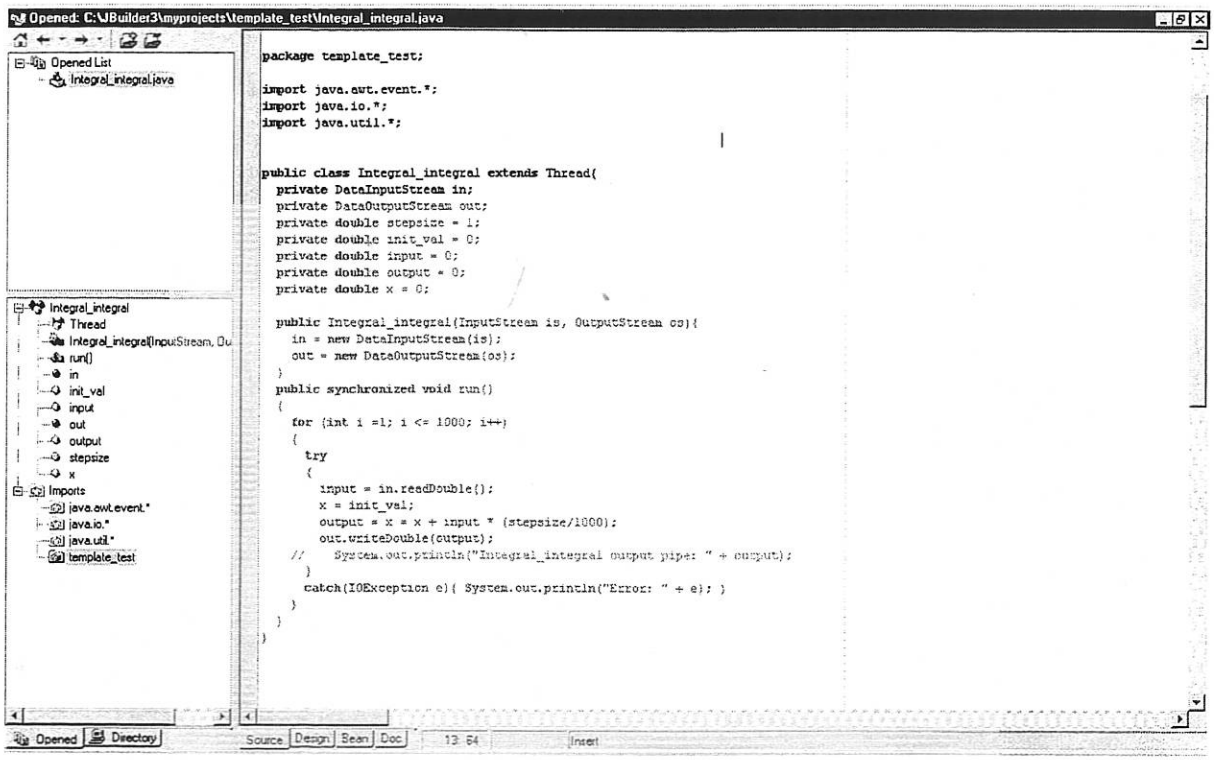

Fig. 6. Java code for a Simulink block.

The following options were considered to develop the automatic Java code generation.

1. Modify an existing scripting file (codeGen.ebs).
2. Use Rational Rose Java code generator and Javadoc.

Using a script file (option one) will retrieve the information from the UML diagrams to create the Java source code. Although, this is simple it requires more development time. Option two involves first generating the Java code using Rose Java tool. Then, it is necessary to create the actual functionality for the application. To create this functionality such as multi-processing communication code and the import of the Template Library as well as its generation, Javadoc seems a straightforward way to do it. Using a few doclets, apart from obtaining classes, methods, fields and tags information, it is possible to create and extend this functionality. Figure 7 shows this generation of code using Javadoc and Rational Rose.

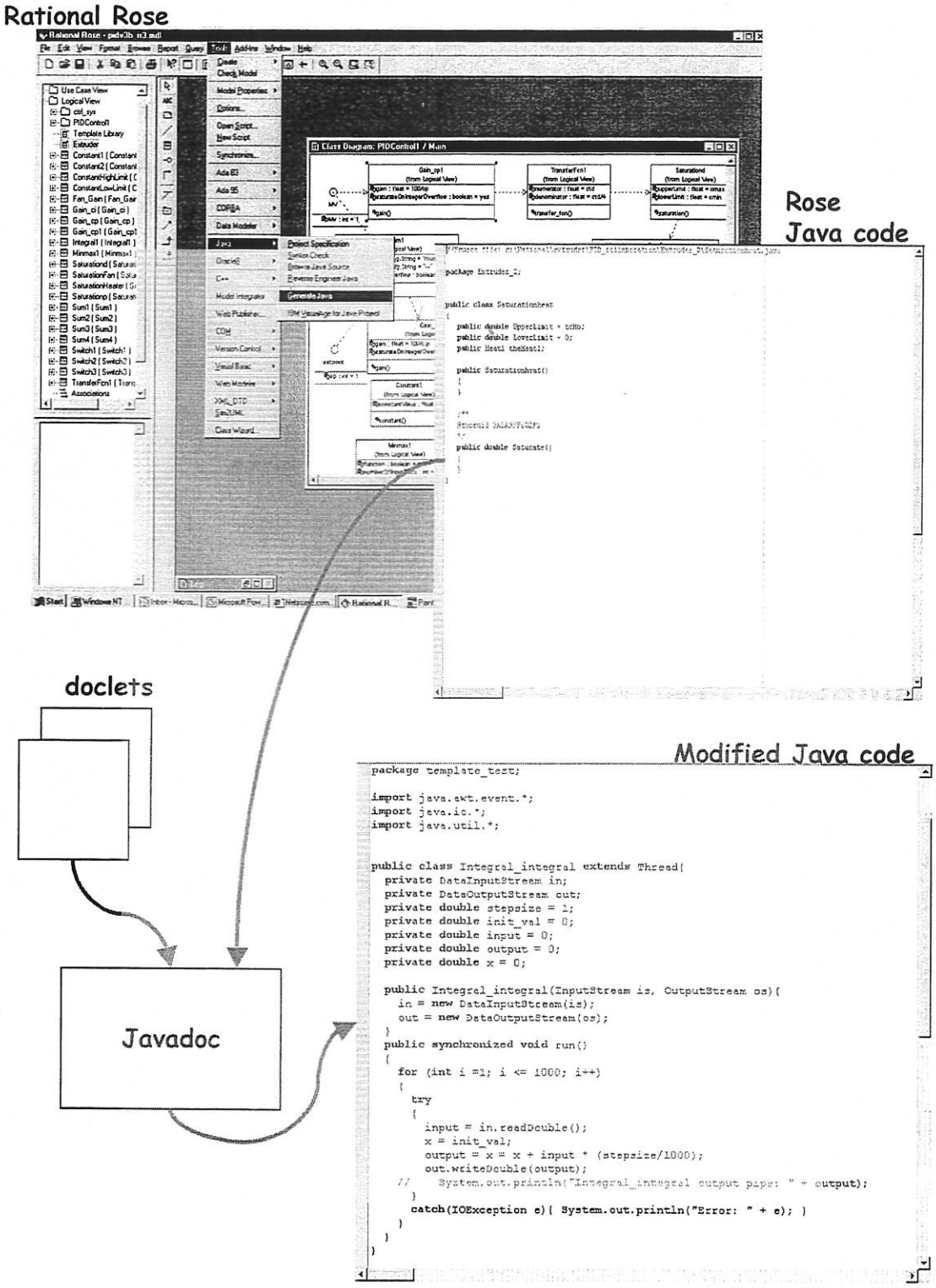

Fig. 7. Code generation using Javadoc and Rational Rose.

Javadoc

- It is used to create the HTML-format Java API documentation
- It is part of the Java SDK
- Its output could be extensible through doclets

Doclet API

- It could access Java class description, tags, and write output to a file
- Simple Java classes could serve as a simple metadata description for code generation

Advantages of Javadoc

- It is not necessary to write any parsing code. This is performed by Javadoc
- Custom Javadoc tags add flexibility
- It supports primitive type keywords
- Java metadata classes can be checked for syntax and other errors by compilation

A similar solution to Javadoc (Doclet API) for manipulating class descriptions is XMI toolkit API. XMI is a full-blown description of UML using XML (eXtensible Markup Language) and it has been used as an exchange format between UML tools (Pollack, 2000).

For the automatic Java source code generator, both options were evaluated. These are the advantages and drawbacks for each option:

1. *Modify an existing scripting file (codeGen.ebs)*
   -This is relatively easy, however it requires more development time.
   -Script language is more limited than Java.
2. *Use Rational Rose Java code generator and Javadoc*
   Step a. This involves generating the Java code using Rose Java tool.
   Step b. To create the actual functionality for the application using Javadoc through doclets.
   -Javadoc doesn't obtain any values of fields, classes or methods.
   +Javadoc is straightforward to generate code automatically.
   +Use of Java language to extend code functionality.
   +The majority of CASE tools supporting UML will provide Java tools to generate code, using a custom doclet will take the information necessary to create Java code with the custom specifications without re-designing again the automatic code generation software.It was decided to use the Rational Rose Java code generator and Javadoc approach, which has shown potential to generate automatically the source code for a Simulink model. Currently, the Java source code for a small controller 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. An application of cable extrusion control has been described where an object-oriented approach is needed to deal with the interaction between control blocks and to map hardware components onto objects. The IDN environment is based on the UML methodology and uses Rational Rose Software. The integration of Simulink into the IDN was

described and further work will include enhance the automatic Java code generator and the integration of the Stateflow and IEC 1131-3 notation to refine and complement the modelling of the target application.

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.

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.

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.

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.

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.

Simulink (1999). *SIMULINK, Dynamic system simulation for MATLAB*. Using Simulink Version 3. The MathWorks Inc.

Stateflow (1999). *STATEFLOW for use with SIMULINK. User's guide*. Version 2. The MathWorks Inc.

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.