1

Infrastructural Software for Model Driven Distributed Manufacturing Systems

Ian Coutts, Marcos Aguiar and John Edwards
Loughborough University of Technology
Manufacturing Systems Integration Research Institute
Loughborough, Leicestershire LE11 3TU - England.
Tel.+44 1509 228250, Fax +44 1509 267725
Email I.A.Coutts@lut.ac.uk, WWW http://msiri.lut.ac.uk

Abstract

The modelling of manufacturing systems has become a necessary part of the drive for manufacturing enterprises to remain competitive within a global marketplace. Manufacturing models are increasingly being used not just as a means of articulating system design but also to drive manufacturing systems at run time. To achieve model execution within a distributed and integrated manufacturing system a number of infrastructural software elements are required.

1 INTRODUCTION

When preparing a paper for this workshop the authors attemped to contribute 'something from the real world'. The following section is a description of a typical industrial problem domain, for which researchers in the application of IT to manufacturing, are deriving solutions in the form of methodologies, software tools and supporting models. This description provides a context against which the CASE tool and infrastructural software described in the paper can be considered.

2 A TYPICAL INDUSTRIAL PROBLEM DOMAIN

In the electronics manufacturing industry the design of fine line printed circuit boards (PCB) implementing highspeed logic designs is a complex problem involving groups of personnel with a variety of skills (Berri, 1994). The process involves the generation of PCB layout designs, their simulation and analysis, followed by inevitable redesign, in an iterative cycle.

I. Jelly et al. (eds.), *Software Engineering for Parallel and Distributed Systems*

This environment of continual evolution requires the discipline of identifying and controlling design versions, in order to maintain integrity and traceability. Figure 1 describes a scenario

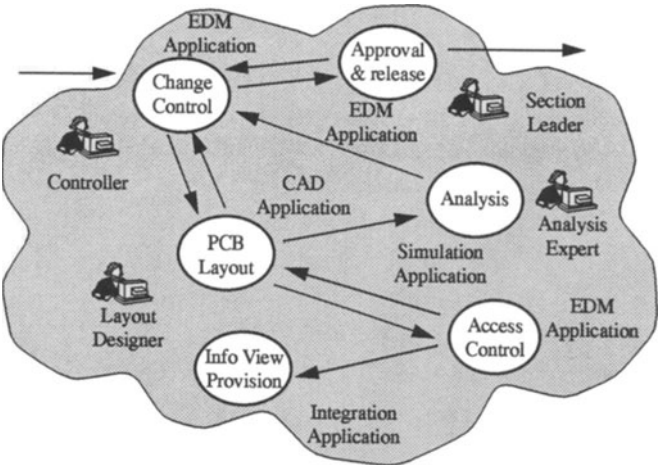

Figure 1. An Industrial Application Scenario

where personnel involved with engineering data management (EDM) must combine with those involved in the design and analysis processes using software applications from a range of vendors.

Required change to a PCB design could be initiated from an external request for a new variant of an existing product. A change control application driven by an engineering department operator could initiate a PCB layout change following completion of the electronic logic design. Following modification to the PCB layout, the company analysis expert will use a range of simulation applications to check for problems such as ringing, crosstalk and EMC (Berri, 1994). Throughout this process file access is controlled by another EDM application, while global information access is enabled through the use of an information view provision application (Weston, 1994). Following successful completion of the new PCB layout the design will be approved and released for prototype production by a senior member of the Engineering Department.

In order to provide a flexible integrated manufacturing software system to support this type of operation there is a requirement to model and rapid prototype a system based on a coordinated set of distributed software objects, where these objects interoperate through message passing. This formal approach to manufacturing system creation is part of a process often described as manufacturing enterprise engineering.

3 AN OBJECT-BASED BOTTOM UP MODELLING TOOL

The Bottom Up tool is intended to be used to model systems by combining manufacturing resources, while separating system behaviour from system function. Behaviour is captured using petri-net models and is executed at runtime, this behaviour is distributed as it is

described within the component objects of a system.
Figure 2 outlines the structure of the object-based CASE tool which was built using IPSYS

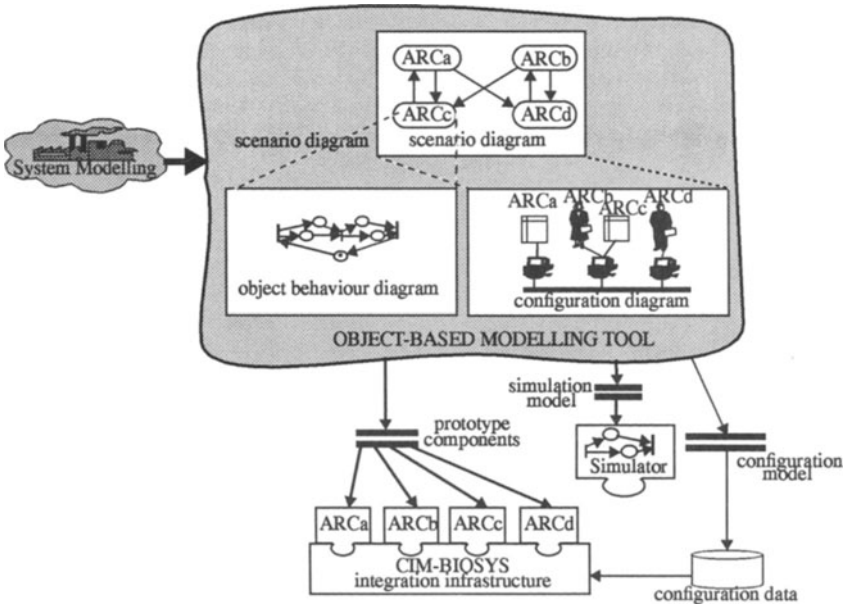

Figure 2. Object-based bottom up modelling tool

Meta CASE technology (Alderson 1991), the figure also shows the main elements of code that the tool can generate. Two aspects are of particular importance in this figure, namely: the design process embodied in the diagrams that the CASE tool supports, and; a capability for generating a rapid-prototype of a particular design which can be executed upon a layer of software which forms an integration infrastructure.

The modelling method encapsulated by the CASE tool comprises three diagramming techniques namely: a scenario diagram (which identifies the major objects (i.e. Active Resource Components* - ARC) which constitute the main components of a system being modelled, and defines the flow of messages among these objects), an object behaviour diagram (which defines the expected external behaviour of an object as perceived by the objects with which it interacts. Such a description utilises a predicate-action Petri-net (David, 1994) to define the internal sequence of actions within the object, as well as the relationships between such actions and external interactions with other objects to which the object in question relates) and a Configuration diagram. (which defines the computer configuration of the system

*. An active resource component identifies a component of a system which is able to execute an element of functionality on its own. It can also be a modelling description which characterises either a human being, an application program or a machine that possess a computerised controller. In the case of this paper, particular interest is placed upon active resource components which characterise software objects. The term Active Resource Component is defined within the CIM-OSA Reference Architecture (ESPRIT/AMICE, 1993)

i.e. on which host computer each active resource component will be executed).

These three diagrams constitute a minimal modelling facility for describing a system comprised of a number of objects which interact with one another in order to perform a common task. Such a description focuses on capturing the behavioural aspects associated with the way in which interactions occur within the system. The actual functionality performed by each object is defined in close association with the predicate-action Petri-net which defines its internal behaviour. This functionality comprises a set of object methods or object member functions which within this paper will be referred to as methods.

4 MODEL CREATION

As stated earlier, in order to engineer an agile* and integrated, distributed manufacturing software system there is a requirement to model and rapid prototype a system based on a coordinated set of distributed software objects, where these objects interoperate through message passing. The methodology embodied in the bottom up CASE tool can provide support for this process. The level of support can be illustrated by examining a general scenario. Figure 3 shows a representation of such a system, modelled via an object-oriented

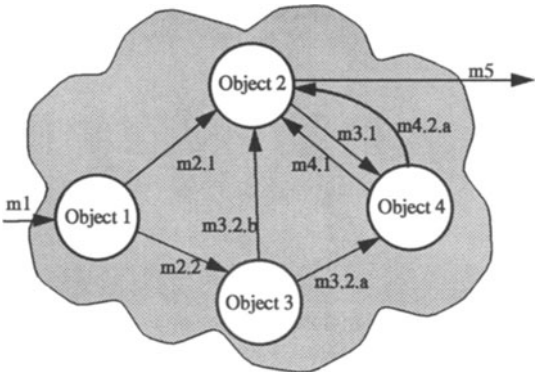

Figure 3. Scenario diagram of a hypothetical system

representation, where four objects are required to coordinate in order to support a distributed manufacturing process.

The objects depicted in Figure 3 perform tasks defined by their internal functionality where these are triggered by messages. This is described by the behaviour diagrams populated with the predicate-action Petri-net models as shown in Figure 4, this functionality may, of course, involve the actions of a human operator using the software object. Figure 4 also illustrates a proposed notation that explicitly classifies the type of action associated with the firing of each transition.

---

*. An agile manufacturing system embodies a high degree of long term flexibility. This provides support for system update and change which enables the manufacturing enterprise to respond quickly to changing market situations.

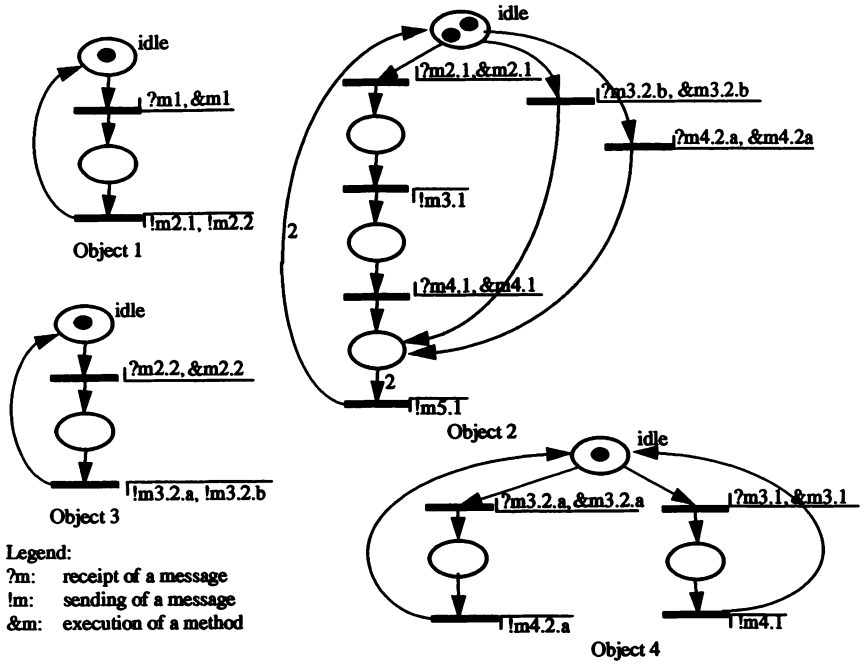

Figure 4. Petri-Net descriptions of the components of the original system

5 CODE GENERATION FOR RAPID-PROTOTYPING

Design information formalised through models produced using the CASE tool are passed in the form of a number of pieces of interpreted code to the infrastructural software that enables model enactment.

This paper now focuses on describing the infrastructural software elements which enable the models produced during system design to be executed during rapid prototyping.

6 MODEL EXECUTION

To facilitate the rapid prototyping of system solutions generated by the CASE tool. researchers at the MSI Research Institute have produced an environment which comprises a set of infrastructural software elements that sit above generally available operating system software. The principal components forming this environment are shown in Figure 5. The figure shows a number of objects interacting via an integration infrastructure, these objects are executable representations of the objects modelled within the scenario diagram of the CASE tool. Each object comprises four distinct areas of functionality as shown in the 'exploded' object in Figure 5, and as described in the following points:

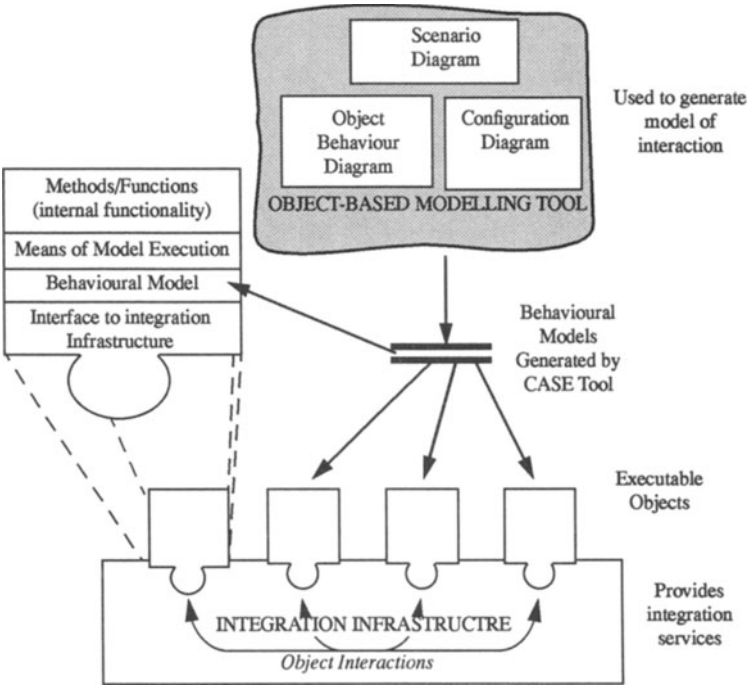

Figure 5. Composition of Executable Objects

- an integration infrastructure interface which enables the object to access the integration services offered by an integration infrastructure.
- a behavioural model which describes the interactions between any one object and all other objects in the system. This is defined within the Object Behaviour Diagram of the CASE tool during the system design phase as a predicate-action Petri-net.
- a number of methods which comprise the internal functionality of an object.
- the means to execute both the object's behavioural model and trigger its internal methods.

The following sections focus on the structure of the objects identified in Figure 5 and the infrastructural software elements used to execute such objects. In order to describe these objects it is first necessary to briefly describe the integration infrastructure on which they interact (a detailed treatment of which is given in (Coutts, 1992)) and to describe the format of the behavioural models which are automatically generated by the CASE tool.

7 THE INTEGRATION INFRASTRUCTURE

An integration infrastructure provides the necessary services to enable interaction between the various software objects which comprise a system. It provides a consistent set of services

irrespective of the objects's physical location, or the operating system and network protocols used. Researchers at MSI have created such an integration infrastructure called CIM-BIOSYS (CIM Building Integrated Open SYStems). The objects achieve interaction by using the integration services offered by the infrastructure. As well as object interaction (message passing), CIM-BIOSYS provides services for file access, data access and system configuration. Services pertinent to this paper include; EST_APP which establishes a peer to peer link between two software applications, TERM_APP which terminates a peer to peer link, STAT_APP, which obtains the status of peer to peer links, SEND_APP which sends untyped data to a connected peer.

The system configuration is produced by the CASE tool via the configuration diagram (see Figure 2). This details configuration information such as which objects reside on which platform, which executable image to invoke for a particular object and which network driver to use to communicate with a particular host. This information is held in a number of ASCII files (produced directly by the CASE tool) which are loaded when the infrastructure is initialised.

8 THE FORM OF THE OBJECT BEHAVIOURAL MODELS

The link between system design (or model building), and model execution is provided by the production of an executable version of the behavioural model created for each object described using the object behaviour diagrams within the CASE tool (as shown in Figure 5). Object behavioural models are represented by predicate-action Petri-nets during the system design phase within the CASE tool and are converted into a textual language (defined by researchers at MSI) during rapid prototyping. It is this textual language which can be enacted to facilitate model execution.

The textual language known as BTL (Behavioural Transition Language) defines predicates to describe interaction with other objects and predicates to execute internal methods. An example textual model for the predicate-action Petri-net Object2 defined in Figure 4 is shown in Figure 6. The language also provides for the representation of global variables which are

```
variable(idle,2).
transition( init1 , ( idle > 0 & recv_object(object1,"m2.1") ) ,
                        ( method(2.1) @ p1 is p1 + 1 @ idle is idle - 1 ) ).
transition( init2 , ( idle > 0 & recv_object(object3,"m3.2b") ) ,
                        ( method(3.2) @ p3 is p3 + 1 @ idle is idle - 1 ) ).
transition( init3 , ( idle > 0 & recv_object(object4,"m4.2a") ) ,
                        ( method(4.2) @ p3 is p3 + 1 @ idle is idle - 1 ) ).
transition( p1-2, ( p1 >= 1) , ( send_object(object4,"m3.1") @ p2 is p2 + 1 @ p1 is p1 - 1 ) ).
transition( p2-3, ( p2 >= 1& recv_object(object4,"m4.1") ) ,
                        ( method(4.1) @ p3 is p3 + 1 @ p2 is p2 - 1 ) ).
transition( p3-idle, (p3 >= 2 ), ( send_object(external,"m5.1") @ p3 is p3 - 2 @ idle is idle + 2 ) ).
```

Figure 6. BTL Model for Petri-Net description of Object 2

used for token counts, arithmetic expressions etc. The following provides some examples of such predicates:

method(2.1) - this executes the method called 'method 2.1'.
send_object(external,"m5.1") - this sends the message "m5.1" to the object called 'external'.
recv_object(object4,"m4.2a") - this tests to see if message 'm4.2a' has been received from the object called 'object4'.

p2 <= 1 - this tests to see if the variable p2 is less than or equal to 1.
p3 is "hello" - this sets the value of p3 to 'hello'.

A behavioural model expressed using BTL consists of a list of transition descriptions comprising a label, a condition and an action, which take the following form:

```
transition(label, condition, action).
```

The label is used to identify a particular transition, but serves no function within a BTL model. If the condition is satisfied the associated action is fired, composite conditions using the operator '&' to denote the AND operation can be used and a list of separate actions can be fired. Every action within the list is executed irrespective of the success or failure of the previous action, here the operator '@' is used to denote this 'follow on' operation. A list of variable initialisations can also be included. If variables are not initialised they are instantiated when they are first encountered and given a value of zero. As an example the following transition

```
transition(init2, (idle > 0 & recv_object(object4,"m4.2a")),
                                        (method(3.2) @ p3 is p3 + 1 @ idle is idle - 1)).
```

is labelled 'init2', and will execute method '3.2', add one to variable 'p3' and subtract one from variable 'idle', if 'idle' is greater than zero and the message 'm4.2a' has been received from object 'object4'.

9 THE STRUCTURE OF THE EXECUTABLE OBJECTS

Figure 7 provides a more detailed breakdown of the functionality within the four principal elements of an executable object. The figure also shows the important relationships between these functional elements. The principal contribution made while researching the requirements for the infrastructural software elements needed to support the bottom up model driven approach are highlighted in the two shaded areas on Figure 7: namely the 'model execution engine' constructed using Prolog* (Clocksin, 1984) and the 'event driven integration infrastructure interface' constructed using 'C' (Kernighan 1978). These two areas of functionality are described in the following sections, but for completeness and to establish the role of each component, the sections below describe each of the four areas from top to bottom as shown in Figure 7.

9.1 The methods which implement the internal functionality of the object.

As introduced earlier in the paper these are well defined and bounded pieces of functionality which perform the operations required for the object to fulfil its purpose. Such methods are identified by the modelling process and the 'trigger points' for their execution are contained within the object behavioural model generated by the CASE tool. As shown in Figure 7the methods are invoked by requests made by the execution engine. The code required to perform these methods is not automatically generated by the CASE tool and must be included by the system implementor. Within the current implementation such functionality can be included as Prolog source code or compiled 'C' object modules. Using the example outlined at the

*. The Prolog selected to implement the executable object was 'C Prolog' (as defined by Fernando Pereira, July 1982, EdCAAD, Dept. of Architecture, University of Edinburgh) this was chosen as it allows extension to both its Prolog environment and the 'C' source code in which it is written.

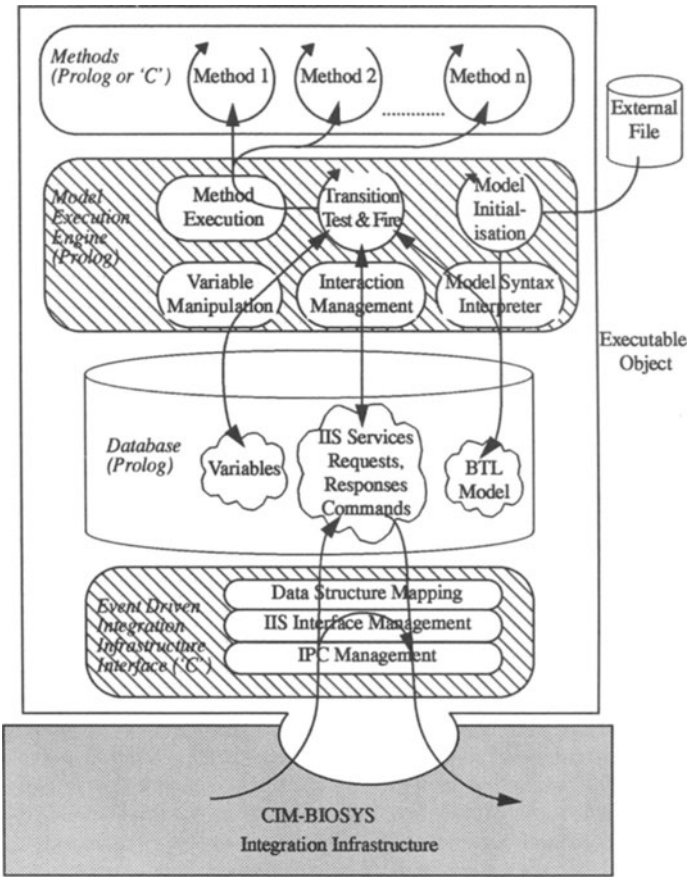

Figure 7. Structure of Executable Object

beginning of this paper, if an object constituted the change control part of a EDM system a internal object method might time-stamp a particular design.

9.2 A behavioural model execution engine.

Executable objects are invoked via the CIM-BIOSYS EST_APP integration service, the name of the file containing the BTL version of the appropriate object behavioural model is supplied at this time. The model import and initialisation process then loads the BTL model from the host computer's file system into the object's internal database. If required peer connections are then established and the transition test and fire process is started. This process continually tests the conditional part of all the transitions contained within the BTL model. If the result of a condition evaluation is true the corresponding action or actions are fired. The subsequent execution of predicates contained within the transition descriptions causes the executable

object to either: manipulate internal variables e.g token counts within the behavioural model, execute internal functionality via the predefined methods or interact with its external environment through message passing via the use of integration services provided by the integration infrastructure (CIM-BIOSYS).

As shown in Figure 7 the model execution engine contains the two processes (model initialisation and transition test and fire) explained above and the following four separate functional support modules.

Model syntax interpreter

This provides both access and parsing of the BTL model held within the Prolog database. The transition test and firing process uses this functional support to examine the transitions held within the BTL model. This functionality has been mainly achieved via extensions to the resident Prolog parser.

Interaction management

This provides access to the integration service requests, responses and commands held within the Prolog database. The transition test and firing process can use this functional support module to issue integration service requests to the database, these will then be sent to the infrastructure by the integration infrastructure interface. The opposite is true for incoming messages, they appear via the integration services from the integration infrastructure into the database and then onto the execution engine.

Variable manipulation

This provides access to and manipulation of the global variables defined within BTL. These variables are manipulated by operators defined by the Prolog language. However the scope of the variables does differ from those defined by the Prolog language, as once they are initialised they can be shared and manipulated by all predicates and transitions until they are explicitly removed i.e they are global in nature.

Method execution

This enables the transition test and firing process to invoke the predefined internal object methods associated with the underlying function of the object.

9.3 A Database.

This serves the executable object as a data repository which holds: a) the BTL model loaded from the external file system; b) any associated variables required to execute the BTL model and; c) integration service requests or responses generated by the model execution engine or the integration infrastructure interface. The database also serves as an interface between the model execution engine and the integration infrastructure interface as both components are driven by the integration service requests held within the database.

9.4 An Integration Infrastructure Interface.

This is an event driven interface which provides the functionality required to allow the executable object to access the integration services offered by CIM-BIOSYS. This interface

also makes the executable object appear as a CIM-BIOSYS compliant application by communicating with CIM-BIOSYS using the required protocol and inter-process communication mechanisms. The interface has been constructed to respond to two different types of event. One event type is the arrival of a data packet from CIM-BIOSYS, typically this could be an incoming integration service command or a response to a service command initiated by the executable object. In both cases the corresponding request is constructed and instantiated in the database. Another example is the arrival of a low level "heartbeat" to check the object is still operating as expected in which case an appropriate reply is issued. The other event type is the instantiation of an integration service request within the database in this case the corresponding integration request is constructed and delivered to CIM-BIOSYS and the request removed from the database.

As shown in Figure 7 the integration infrastructure interface contains the following three separate functional support modules.

Data Structure Mapping

This maps between the form of the structured data packets required by CIM-BIOSYS integration services and that defined in BTL. It also provides the mechanism by which asynchronous access to the database is enabled. This insures incoming integration service requests are processed concurrently with the transition testing and firing process.

IIS Interface Management

This manages the necessary interaction with CIM-BIOSYS through achieving the following: executing object initialisation sequences; by performing any handshaking required during normal operation; by executing the object termination sequence; by the handling of status requests from other applications, and; by the routing of integration service requests between the internal database and CIM-BIOSYS.

IPC Management

This is a facility which uses UNIX Inter-Process Communications to facilitate the exchange of structured data packets between the executable object and CIM-BIOSYS.

The executable objects described within this section provide a means for the execution of behavioural models generated by the CASE tool. Additionally they provide an environment in which a number of implementation features of the whole model execution process can be tested and enhanced. The model execution engine can be readily modified to accommodate changes in behavioural model syntax, behavioural model functionality and integration services used. This flexibility is principally due to the engines construction being based on the Prolog programming language. However, flexibility does not compromise runtime performance or ability to function in a distributed environment due to the event driven integration infrastructure interface being implemented using the 'C' programming language.

10 CONCLUSIONS

This paper has introduced a CASE tool for the design, implementation and execution of integrated manufacturing systems. The paper has focused on a description of the infrastructural

support software which provides enactment facilities for the behavioural models created by the CASE tool. The distributed systems created using the tool and its infrastructural support elements benefit from a model driven approach which provides support for the key manufacturing enterprise need of accurate implementation of user needs and an improved ability to respond to required change.

The CASE tool and the infrastructural software described provide the support which brings together conventionally separate life cycle phases, from 'detailed design' to the 'configuration and execution' of systems upon an industrially tested integration infrastructure.

11 REFERENCES

Alderson A. (1991) Meta-CASE Technology. Lecture Notes in Comp. Science Software Dev. Env. and CASE Technology. Proc. of Euro. Symp. p81-91. Springer-Verlag. Germany.

Berri, J. (1994) High Speed Heaven, Printed Circuit Design.

Clocksin W.F. and Mellish C.S. (1984) Programming in Prolog, 2ed., Springer-Verlag.

Coutts, I. A. et al. (1992). Open Applications within Soft Integrated Manufacturing Systems, Proc. of Int. Conf. on Manufacturing Automation, Hong Kong, ICMA 92.

David, R. And Alla, H. (1994) Petri-nets for modelling of dynamic systems - a survey. Automatica, vol. 30, no. 2, pp. 175-202.

ESPRIT/AMICE. (1993) CIM-OSA Architecture Description, AD 1.0. 2.

Kernighan B.W, Ritchie D.M. (1978) The C programming Language, Prentice-Hall.

Weston RH, Clements P, Murgatroyd IS. (1994) Information modelling methods and tools for manufacturing systems, Proc. Lean/Agile Manufacturing in the Automotive Industries Conf. of the 27th Int. Symposium on Advanced Transportation Applications (ISATA), Aachen, Germany, pp227-234, ISBN o 947719 70 9.

12 BIOGRAPHY

I A Coutts spent two years at Marconi Research as a research scientist, working on industrial assembly automation and robotics projects. He has spent the last seven years as a member of the Loughborough University Systems Integration Group, and currently works in the MSI Research Institute at Loughborough. Particular responsibilities include work on infrastructural software and facilities for enabling model enactment.

M W Aguiar spent seven years as managing director in charge of the Integrated Automation Division of a Research and Development Institute of the Federal University of Santa Caterina/ Brazil. He has spent the last three years as a member of the MSI Research Institute, involved in the 'Model-Driven CIM' project, with particular responsibility for the conception, realisation, application and evaluation of SEW-OSA.

J M Edwards gained his Ph.D from Loughborough University in 1994. Having spent 13 years in UK process and manufacturing industry, being involved in the creation of computer control and information systems, he joined Loughborough University in 1987. During his 8 years at Loughborough he has been involved with the Systems Integration Group and is now a member of the MSI Research Institute where his role is as principal investigator on a number of UK government funded research initiatives.