10

Case Tools for Flexible Manufacturing Systems

Prof. Dr.-Ing. Dr.-Ing. E.h. M. Weck, Dipl.-Ing. J. Friedrich, Dipl.-Ing. Th. Koch, Dipl.-Ing. R. Langen
Laboratory for Machine Tools and Production Engineering (WZL), Aachen University of Technology, Germany

Abstract

Case Tools for application engineering in the field of flexible manufacturing systems (FMS) have gained importance due to the rising complexity of the control software. This paper presents three different case tools from the current research activities at WZL which are part of COSMOS, an open control architecture for flexible manufacturing systems. *CellDesign* is a graphical development tool for cell controllers using a petri net based approach. *CASCADE* provides a framework with class libraries and design patterns for an object oriented development of shop floor applications. *MMS-3D KIT* is a set of different Case Tools for the development of device drivers which are needed to integrate machine tools incompatible with the Manufacturing Message Specification (MMS).

Keywords

Case Tool, Cell Control, Class Library, Design Patterns, Development Environment, MMS, Open Architecture, OSI, Petri-Nets, Shop Floor Control, Software Engineering, Workframe

1 INTRODUCTION

Enterprises, large and small, are more and more organized into manufacturing cells to simplify business control, enable lean production and increase the efficiency of production life cycle. These changes require new control systems based on new technologies such as open systems, client server architectures and object orientation which offer flexible and efficient solutions. The major hurdles in entering this new world are related to software: the time to develop it, the ability to maintain and enhance it, the limits of a program's complexity with regard to its maintainability and the time it takes to become familiar. This leads to the major issue information systems have today: long time to market, insufficient quality, high cost and lack of interoperability. While hardware cost are decreasing, software expenses are still rising.

A. Storr et al. (eds.), *Software Engineering for Manufacturing Systems*

In this paper new and powerful Case Tools used to develop software for the COSMOS control architecture for flexible manufacturing systems (FMS) are presented (Figure 1). COSMOS is intended to provide efficient, coherent and cost effective control of manufacturing process at the factory level (Weck 1995) by providing a generic and open control architecture. It is based on an integrating infrastructure linking the diverse control applications into an integrated, but distributed system.

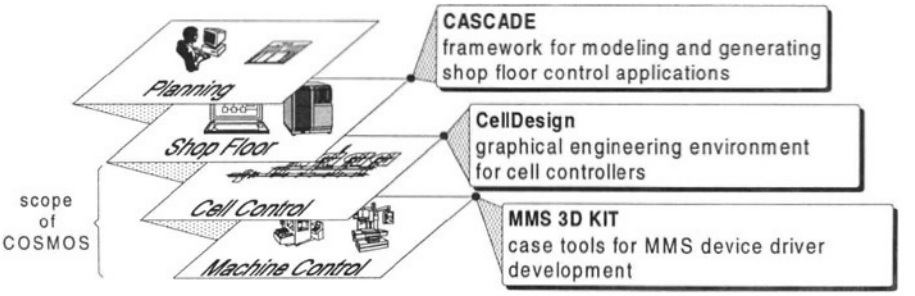

Figure 1 Case Tools of COSMOS.

Due to the complexity of flexible manufacturing systems the development of control software requires a systematic engineering approach to improve software quality and reduce development time and cost. Therefore, user oriented software engineering tools are an essential part of the COSMOS architecture, e.g.:

- **CASCADE** is a development environment consisting of several tools for modeling and generating *shop floor control applications*. The tools of CASCADE support an object oriented approach which is based on two different types of objects: resource objects and dynamics objects. The application framework provides base classes and design patterns for these objects.
- **CellDesign** is a graphical engineering environment for *cell controllers*. It combines the power of petri nets with the benefits of reusable software. While basic functions of a cell controller are provided by so called Function Objects, applications are designed by glueing these functions together graphically.
- **MMS „3D" KIT** are diverse Case Tools needed for the *MMS device driver* development. The MMS 3D-Kit enables the integration of machine tools using communication standards other than the International Standard Manufacturing Message Specification.

COSMOS provides a powerful software architecture that enables FMS control systems to be defined, developed and installed in the minimum time and with the maximum possible reuse of existing components.

2 CELLDESIGN - A CASE-TOOL FOR CELL CONTROLLERS

Due to the complexity of Cell Controllers software development requires a systematic engineering approach to improve software quality and to reduce development time and costs. Though many CASE tools are available today they are often not suitable for the development of cell control software. This is because they either do not provide powerful reuse concepts or they do not support modeling of event-driven systems with lots of parallel execution, but interdependent processes (Frey, 1992). For that purpose a specific CASE tool called *CellDesign* has been developed at WZL.

2.1 Basic Concepts of CellDesign

The client-server concept of COSMOS provides a mechanism for the development of cell control applications based on reusable software. It is suggested that control applications may only contain the control flow of a program while the function execution itself is dedicated to servers. A function call in an application causes a message to be sent to the corresponding server (Figure 2). This approach has two major advantages. Firstly, application modeling is independent from function implementation. Secondly, services (functions) provided by existing servers are reusable. As a consequence powerful and generic server libraries help to increase the reusability and to decrease the implementation effort.

In cell control applications sequential data processing, algorithms and control of dynamic processes can be distinguished. For modeling of transaction oriented and algorithmic applications, methods such as flow-charts are available. Unfortunately these methods are not suitable for the design of parallel, dynamic and asynchronous processes, which are characteristic of manufacturing cells. Therefore a modeling approach based on so called *action-nets* has been developed. The basic idea is that applications are modeled as objects which react to external events. For example, an event called "new cell order" is sent by a shop floor control system to the cell controller, as shown in Figure 3 . Such an event is specified by

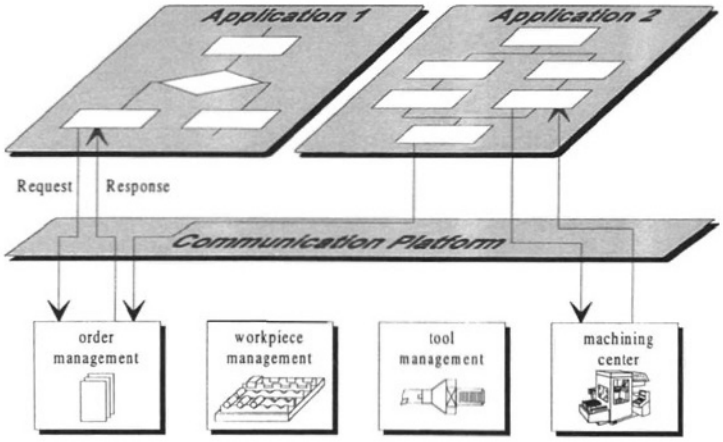

Figure 2 Separation of control flow and function examination.

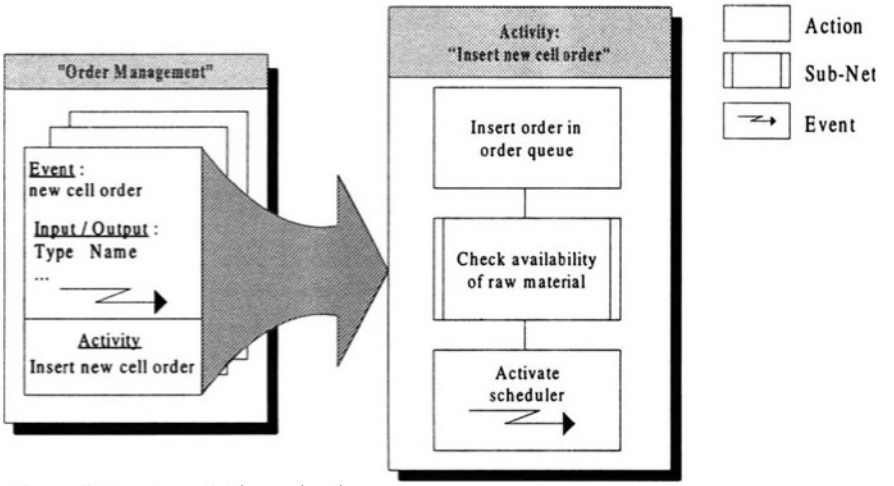

Figure 3 Events, activities and action nets.

a unique identifier and a list of input-data. An event causes an *activity* of the application which can be described in more detail by a sequence of *actions*. Actions are either atomic functions executed by servers or further activities in the sense of "sub-nets".

Modeling of control structures in action-nets is based on mechanisms provided by petri-nets.

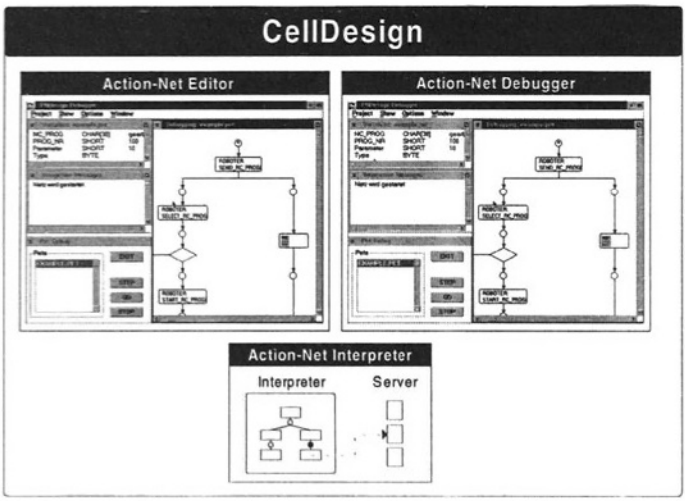

Figure 4 Integrated Development Environment of CellDesign.

A state of a petri-net can be considered as a precondition for the execution of an action which is represented by a transition. An action will be executed only if all preconditions are fulfilled respectively marked. With petri-nets it is very easy to model control structures like sequence, loop or parallelity and they also simplify synchronization between parallel processes.

Based on these concepts for application engineering a CASE tool called *CellDesign* has been developed at WZL, which supports the overall process of application design and development. As shown in Figure 4, this CASE-Tool consists of three components: action-net editor, action-net debugger and action-net interpreter. The editor is a powerful, fully graphical oriented tool for action-net design. It provides all basic elements like states, actions and connections as well as elements which represent start and end of an action-net.

At the end of the design phase, the code generation process must be activated. This process compiles all action-nets into a formal language, which can be interpreted by the run-time system or by the debugger. For debugging the graphical net representation will be used to ensure that a system developer is able to test an application at the same level of abstraction and representation as used in the design phase.

The execution process of an action-net is represented by marks flowing through the net. In "step-by-step" mode the execution of action-nets is under full control of the developer. Furthermore it is possible to monitor all variables of an action net.

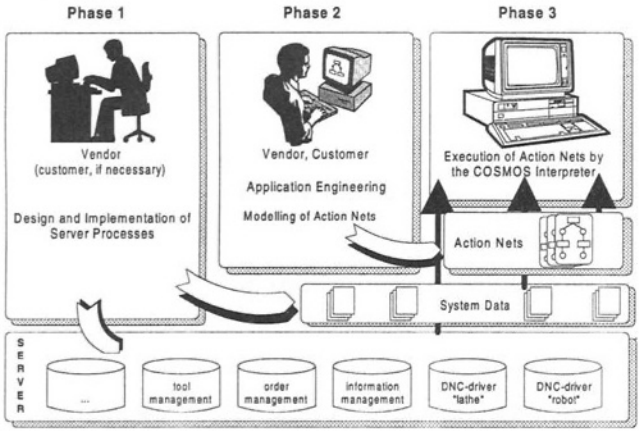

Figure 5 Phases of Application Engineering in CellDesign.

Figure 5 summarizes the engineering approach for the development of end-user specific cell controllers based on reusable software. In phase I software vendors define basic and reusable functions combined in servers which represent the "core functionality" of a cell controller.

In phase II the action-net editor will be used for end-user specific application design on top of the existing "core functionality". Functions which are not available have to be implemented separately. At the end of phase II the code for application execution will be generated automatically, so that there is no more implementation effort for the developer in phase III.

3 CASCADE - A CASE-TOOL FOR SHOP FLOOR CONTROL APPLICATIONS

Looking at recent developments in shop floor control systems there is an increasing demand for systems offering a higher degree of adaptability. The first generation of shop floor control systems had a monolithic structure (Kohen, 1986) which was superseded by a second generation of systems having a modular architecture (Lange, 1993, Pritschow, 1991). The adaptability of these systems is based on the exchange of entire modules. However the possibilities concerning adaptation do still not readily facilitate individual solutions. Object-oriented technologies offer promising possibilities.

According to Zipper (1994) ease of use, flexibility and the capability of being integrated are the most important requirements for shop floor control applications. These features are necessary to guarantee long-term use. Therefore the following requirements on the design of a class library for shop floor control applications should be fulfilled:

- separate modeling of the dynamics processes and the classes itself
- low degree of links between classes
- simple applicability
- extendibility / maintainability
- flexibility / universality
- stability

The application framework developed for CASCADE predefines the architecture of the software system by fixing the structure and interaction of the single components (Booch, 1994, Pree, 1994). This framework approach does not only support reuse of software at the source-code level and the domain specific class hierarchy. In addition it supports the reuse of the entire system-architecture. Therefore a rudimentary, application independent executable skeleton is available. It can be adapted to individual demands by modifying the application specific „hot spots" (Pree, 1994). For this purpose the user is supported by a graphical tool set as described above. This allows a high degree of reuse, because not only the class library (based on design patterns) (Pree, 1994, Gamma, 1994) can be reused, but also the „glue" between the components, (e.g. exception handling).

3.1 Modeling approach

The modeling approach supported by CASCADE, a toolset for modeling and generating shop floor control applications, is based on two types of objects, resource objects and dynamic objects. Real objects physically existing in the production system are modeled as resource objects - sequences and strategies of material flow etc. as dynamic objects. Resource objects describe objects such as machine tools, transportation units, buffers, tools to be supported by the software system. They are modeled applying the notation as presented by Rumbaugh (1991), and consist of attributes, methods and a state model. Instead of identifying an object's state by one variable (LatheState = {WORKING, DEFECT, ...}) the state can be described by any combination of the object's attributes. Dynamic objects incorporate the interaction of resource objects. Processes within the manufacturing system are described in this type of object. Dynamic objects are modeled using scenarios describing these processes. The reason for introducing two types of objects is that manufacturing systems differ mainly in the

processes to be carried out within the system. Therefore primary modifications for adaptation can be carried out in the dynamic objects.

3.2 Development environment

CASCADE is a development environment consisting of several tools for supporting a user during the phases of modeling and application generation by applying the above approach. A *repository* is the central integrating unit for the tools, serving as the storage for data generated during the design and development phases. It is implemented using an object-oriented database on an OS/2 platform. Basic tools are the *Resource Object Builder* and the *Dynamics Object Builder* (Figure 6).

They are needed to build the model of the system according to the approach described above. The design and development cycle is stored in the repository. Afterwards a syntactic check can be carried out using the *Model-Verifier*. If the verification has worked successfully the software based on the model can be generated automatically using the *Application Generator*. A *Documentation Generator* enables the automatic extraction of information out of the repository and the generation of a paper-based documentation using a standard text processor.

The core of CASCADE is a class library which provides base classes and patterns to build resource and dynamic objects. The basic concepts of this library will be described next.

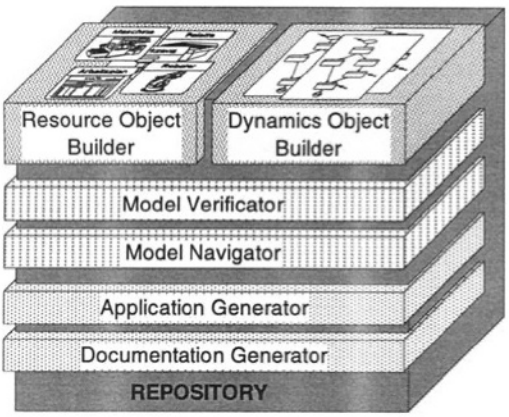

Figure 6 Architecture of the Development Environment.

3.3 Basic concepts of the application framework

The class library designed for the development of shop floor control applications is implemented in C++ on an OS/2 platform and includes base classes with extensive features. Unlike other approaches for class libraries for shop floor control applications (Schmid, 1994) it includes fundamental concepts for software development. Therefore it not only provides classes for items like machine tools, buffers etc. The underlying concepts are important because they are the basis for the design and development of the entire class library.

Basic concepts include runtime type information (RTTI) (Chen, 1995; Stroustrup, 1992) as well as exception and contract handling. Other concepts which have been implemented are association, iterators, base classes for the application itself, the main program, dynamics objects, i/o-interfaces (e.g. terminals) etc. Furthermore, we distinguish between active and passive objects. Active objects have their own thread on a certain site e.g. an object representing a unit in the production system. Passive objects are used by active objects and represent items like tools and fixtures. Classes needed to give objects activity (including communication and distribution) have also been realized. The following paragraphs give a brief summary of some of the basic implemented concepts.

RTTI allows a dynamic (at runtime) type-checking in C++. Regular C++ only features static type-checking. RTTI is needed to avoid runtime errors using „containers", which are often used in shop floor control applications. Design by contract is used in several OOD-methods (e.g. Wirfs-Brock (1993)). This concept is also not supported by regular C++ (unlike Eiffel - see Meyer (1994)). To improve system stability and robustness this feature has been implemented in terms of precondition, postcondition and invariant. A precondition tests whether the client keeps the contract at the beginning of a method. A postcondition has to be valid after carrying out a method to enable a server to keep its contract. An invariant is a condition which should always be true. In combination with the implementation for the design by contract, an extended warning and error handling mechanism has been implemented using several error levels. An error carries out a *throw()* statement while a warning does not. Both statements put information in the logfile. However a program can continue its regular process after a warning occurred. An exception needs an error handler. For dynamic objects suitable „handlers" are available. For resource objects they have to be implemented by a developer depending on the application.

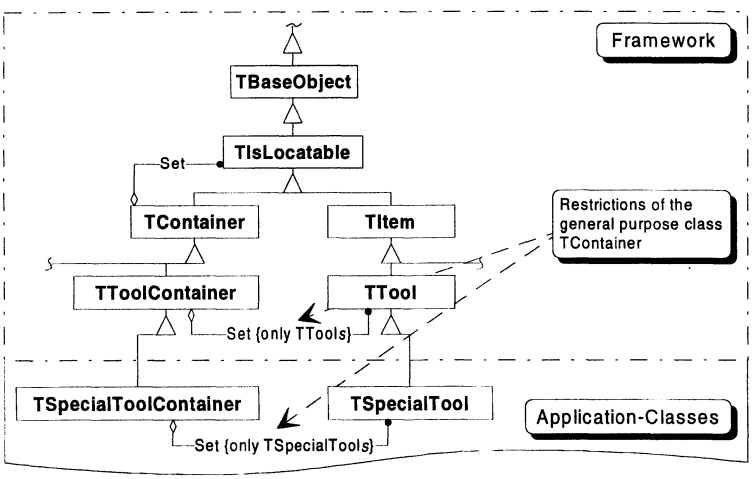

Figure 7 Extract of the provided class library.

An association is one of three types of relations between classes and is used extensively in OO-methods. In shop floor control applications they are needed for example, to design the

relationship between order and customer. An association could be implemented using pointers. However pointers are often used without modeling „real world" relationships. According to Rumbaugh (1991) associations should be implemented as bi-directional. For this reason a base class TIsAssociatable has been developed which allows one to build relationships between objects in the same sense as in the design.

To step through any type of container or list, a generic iterator concept has been implemented. Applying iterators allows uniform access to data, in addition to encapsulating the needed mechanisms and hiding them from the user.

Figure 7 shows a brief extract from the class library. The framework provides generic application classes like TToolContainer or TTool. TContainer and TItem are derived from TIsLocateable, which describes objects having an identifiable position within the system. TContainer itself can contain any type of locatable object. The class TToolContainer is derived from TContainer. However a restriction has been made on the types of objects this container can carry. Only objects belonging to the class TTool can be taken by instances of TToolContainer.

The mechanism needed to build this „special" container is shown in Figure 8. The method *Admit()* of the base class TContainer uses a precondition to test the object to be put into the container. The precondition uses the method *AdmissionTest()* to verify whether the object is of type TTool. This is carried out by applying RTTI (method *IsA()*).

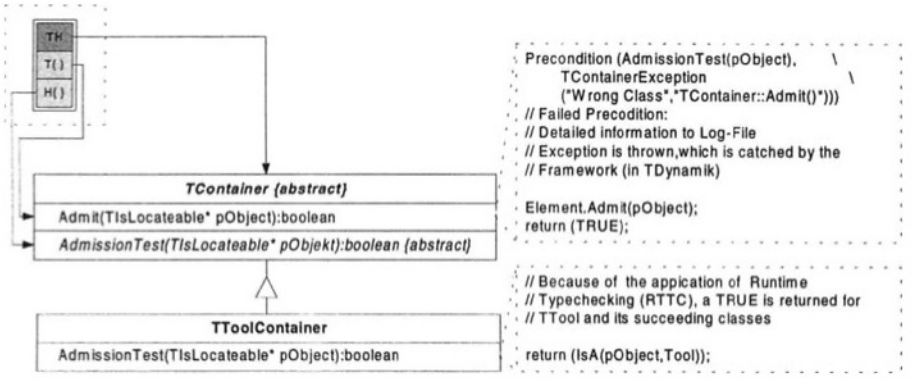

Figure 8 Building a Special Container using the Base Class TContainer.

If a user wants to realize a restriction to TSpecialTool, the only modification needed is:
boolean TSpecialToolContainer::AdmissionTest(TIsLocateable pObject)*
{ return(IsA(pObject, TSpecialTool)); }

This brief example outlines some of the possibilities the class library offers to a developer. In a similar way patterns (e.g. for tool assembly plans, setting up a system layout etc.) have been implemented. These base classes enable a user to develop the resource classes in a natural way. In combination with the state model the needed flexibility for building resource objects is given. The dynamic behavior of the system can easily be changed by adapting dynamics objects.

4 THE MMS „3D" KIT: AN MMS BASED APPROACH FOR INTERACTIVE DEVICE DRIVER DEVELOPMENT

In order to accomplish the electronic information interchange between automation devices like machine tools, robots, etc. on the one side and control computers as well as DNC hosts on the other side, the control computers and DNC hosts have to be equipped with dedicated software modules. These software modules normally are called „device drivers". A device drivers task is to adapt the specific communication interfaces and communication behaviors of a specific automation device to the control computers or DNC hosts software.

Device drivers are commonly implemented as independent software modules having two different interfaces for the information exchange.

The first interface is the interface to the controlling and DNC manufacturing application software modules. To avoid interdependencies between the manufacturing application software and the device driver software modules and to avoid software customization, the device drivers need to be implemented in the same manner for all the different automation devices in the manufacturing environment.

The second interface of a device driver is used to exchange information with the remote controller of the automation device under control of the manufacturing application software. Here the information exchange is carried out via the different current device specific communication platforms.

For the various available automation devices nowadays the device drivers have to be specifically created. This is caused by the high number of different vendor specific communication protocols which were developed in parallel in the past as a result of missing communication standards.

This situation increases the implementation efforts and costs in a way which cannot be accepted by industrial system users. Therefore in recent times a large financial expenditure has been committed to the international standardization of vendor independent generic communication protocols for automation devices. Today as a result of these activities international standards are available which for example allow remote data base access (RDA), remote file handling (FTAM) and the remote control of automation devices (MMS with its associated Companion Standards).

The implementation of these standards in device controlers will decrease the development expenses because necessary software customizations in the manufacturing application software can be significantly reduced.

With MMS -that is the Manufacturing Message Specification- since 1990 an international standard is available solving the communication problems in manufacturing environments. But due to the current global situation in the metal-processing industry and due to the complexity of MMS controller software implementations, MMS communication interfaces have not found the wide spread use that they should have until now. Even today latest machine tool control developments are offered with a communication technology based on obsolete concepts.

But there is no serious doubt that MMS compliance will lead in the long term to automation devices with generic vendor independent communication interfaces (ESPRIT CCE-CNMA, 1995). Because of the above mentioned reasons and keeping in mind that most of the devices used in industrial manufacturing environments today are not equipped with MMS interfaces, at the Machine Tool Laboratory of Aachen University (WZL) a kit for graphical interactive device driver programming is under development (Figure 9). The kit is called the „MMS 3D

kit" and is one of the results of WZL's research activities regarding generic device driver architectures. This architecture should allow one to create device drivers which fulfill the MMS requirements, which are independent of computer operating systems, which drive devices with vendor-specific communication interfaces and which provide a generic interface to the manufacturing application software.

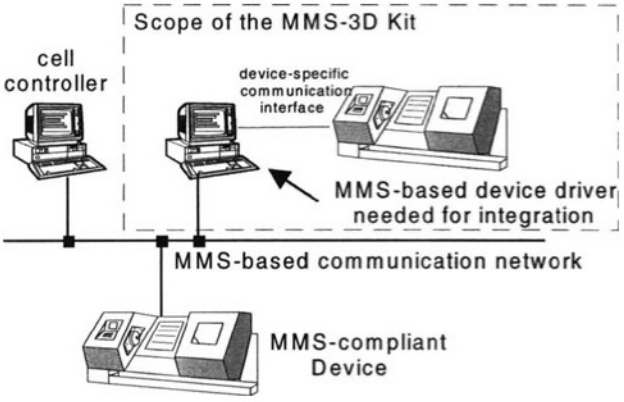

Figure 9 Scope of the MMS-3D Kit

4.1 Kit Profile (Functionality)

4.1.1 Communication Interface between device driver and manufacturing application software

In an object oriented way MMS has specified 11 different communication relevant object classes and 87 accompanying communication services. Generally, with a small subset of the provided MMS objects and services described in so-called MMS Companion Standards, the communication behavior of any automation device can be modeled in a generic way (Friedrich, 1992). Mainly the MMS object classes VMD (Virtual Manufacturing Device), Variable, Domain, Program Invocation and Event Management are needed.

The MMS 3D Kit provides modeling tools for the mapping of vendor-specific communication behavior to dedicated MMS communication objects and services. For this purpose the device drivers developed with the MMS-3D Kit apply the following MMS objects and services.

The VMD services Status, Identify and GetNameList are supported. The service Status allows one to check whether the device is ready for use or not. The Identify service makes it possible to get detailed information about the connected device. The names of all MMS objects implemented in the device driver can be requested with the GetNameList service.

The MMS-3D Kit permits mapping of Control-specific variables to MMS variables, which can be read or written with the corresponding MMS services. Furthermore the MMS-3D Kit permits the management of domains (e.g. NC/RC programs) within the device driver.

All Program Invocation services offered by MMS are supported by the MMS-3D Kit. In addition to that the device driver provides complete program management.

The device drivers permit the feature of the remote creating and deleting of event objects via a communication channel. With the support of these objects other MMS objects like variable objects can be monitored. If for example a user wants to be informed about the change in a specific variable value, the event objects pass the appropriate information to the MMS Information Report service.

4.1.2 Communication Interface between Device Driver and Automation Device

Today different specific communication protocols are offered by almost any control vendor for the remote control of devices in manufacturing environments. The data packets are usually transmitted with line control procedures like the LSV/2 or 3964/R procedure which are often used as De-Facto standards in the German industry (Weck, 1995).

The MMS-3D Kit allows the modeling programming of the vendor-specific data packets described in the device controls operating manuals as well as the accompanying protocol. Even the programming of vendor-specific line control procedure is possible with tools from the MMS-3D Kit.

At the moment an open architecture for control systems is under development within ESPRIT project 6379/9115 „OSACA". A major output of this work is a communication platform for the interprocess communication within a heterogeneous control hardware is specified. To integrate OSACA control systems into an OSI environment, device drivers are required too. The task of these device drivers is the conversion of both MMS and OSACA protocols and vice versa. The MMS-3D Kit eases the programming of such OSACA-MMS device drivers.

4.2 Basic concepts

The device drivers developed with the MMS-3D Kit provide a generic, MMS conformant communication interface to the manufacturing application software. In the following the processing of an MMS service indication in a device driver is described (Figure 10). The data included in the service primitive of the service indication are temporarily stored in the global data area of the device driver. With the help of the MMS identifier received as information in the service indication, the corresponding MMS communication object is found in the object manager. The communication object that was found has a reference to an accompanying transaction net which is executed. The transaction net undertakes for example the task to send and to receive the packets needed for the communication with the automation device. The construction of the packets is carried out with the help of the global data. After receiving the response packet from the device control, a positive MMS response is issued. Otherwise the MMS response is negative.

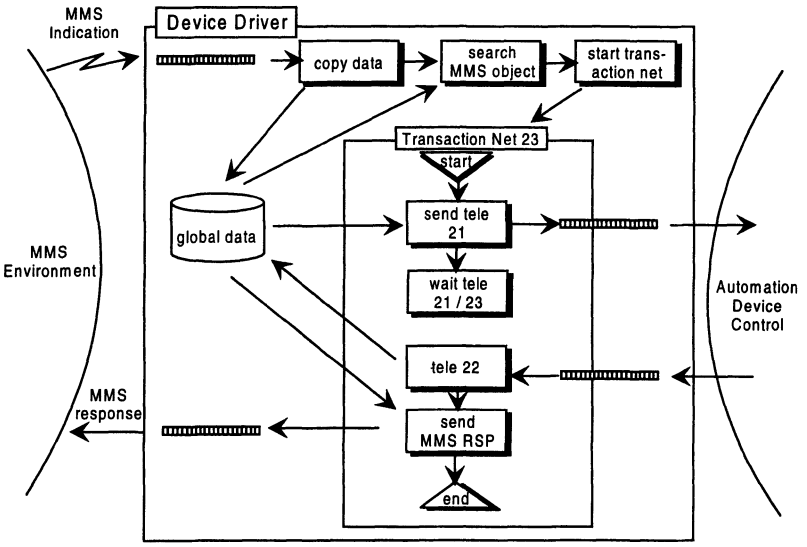

Figure 10 Device Driver Architecture.

4.3 Kit Tools (Editors)

The previously introduced MMS indication processing within the device driver has to be programmed completely. It is the task of the MMS-3D Kit to reduce the actual programming work and the time needed for it and to reduce the programming faults to a minimum. The MMS-3D Kit provides the user six with tools for the development of the different steps needed for indication processing in a device driver. These tools are:

4.3.1 Protocol Editor

The protocol editor is used for the graphic programming of line control procedures. The code programmed with the Protocol Editor is block oriented. Each block has clear pre-conditions, which must be fulfilled, if the corresponding block shall be executed. A pre-condition can be e.g. a specific received sign sequence. Within a block, sign sequences can be sent out via the communication medium or received.

4.3.2 Data Tree Editor

To allow graphically interactive programming of information processing procedures within the device driver, the creation of data structures is necessary. With the help of the Data Tree Editor, data structures can be created, dialogue aided and saved in project files. These data structures can be displayed in a graphic data structure overview and overtaken with a simple mouse click in the necessary processing dialogues.

4.3.3 Data Block Editor

With the Data Block Editor functions, can be programmed to copy data in converted formats from a data stream to device driver specific data structures and vice versa. These functions are mainly used for developing encoding and decoding procedures for packets.

4.3.4 Key editor

The task of the Key Editor is to assign clear recognition keys to specific packets so that an automatic identification for the decoding of the packets and further processing becames possible.

4.3.5 MMS Object Editor

The MMS object editor allows one to create MMS communication objects, which handle the object related information processing within the device driver. For the modeling of internal information processing, transaction nets are used. All the MMS communication objects own one transaction net. The task of the transaction net is to request necessary information from the attached device for further processing in single actions with the help of the device-specific communication medium.

There are differences in the identification of the various MMS object classes. MMS variables created with the editor have an unambiguous identifier and belong to specific local variables in the global data of the device driver. The attached transaction nets of variable objects are specifically for each object. This is different for the MMS object classes Domain and Program Invocation. Objects of these two MMS classes are created with different identifiers during run time and not during the programming of the device driver. Therefore the related transaction nets are common for each class and not for each instance.

4.3.6 Transaction Net Editor

Protocol sequences can be programmed graphically and interactively with the Transaction Net Editor. These transaction nets are assigned to MMS communication classes or to single MMS communication objects. From examinations carried out at the WZL, elementary modeling elements were derived which can be used for the programming. If there is a special modeling element needed, which is not present in the editor, it can be designed with the help of user dialogues.

Concluding the MMS-3D Kit is a migration aid that allows one to program graphically and interactively MMS-based device drivers needed for the integration of manufacturing devices with vendor-specific communication interfaces in OSI-based communication networks. In the future this kind of integration work will no longer be needed and heterogeneity will no longer be a problem if all manufacturing devices are OSI-compliant.

5 CONCLUSION

Up to now there is a significant lack of case tools which are designed and applicable for FMS software. The systems presented in this paper support new approaches to the development of such case tools. CellDesign facilitates the development of customer specific cell control software, CASCADE provides a framework for shop floor control applications and the MMS-

3D KIT enables a user to integrate machine tools incompatible with the Manufacturing Message Specification. The tools presented help to reduce development time and cost as well as the implementation effort of individual software solutions. Future research activities will focus on increasing reusability of software and extending the functionality of the case tools and the provided class libraries.

6 REFERENCES

Booch, G. (1991) Object Oriented Design With Applications. The Benjamin/Cummings Publishing Company, Inc.

Chen, J.B. et.al. (1995) Pursuing safe polymorphism on OOP. in Journal of Object-Oriented Programming, March-April.

ESPRIT Consortium CCE CNMA (1995), CCE: An Integration Platform for Distriubuted Manufacturing Applications. Springer Verlag

Frey, V. (1992) Planung der Leittechnik für flexible Fertigungsanlagen. Dissertation Universität Karlsruhe.

Friedrich, A. (1992) Offenes DNC-kommunikationssystem für numerische gesteuerte Arbeitsmaschinen.Reihe 20 VDI Verlag

Gamma, E. et.al. (1995) Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley.

Kohen, E. (1986) Adaptierbare Steuerungssoftware für Flexible Fertigungssysteme. Dissertation, Aachen University of Technology.

Lange, N. (1993) Dezentrale universelle Steuerungsarchitektur für Flexible Fertigungssysteme. Dissertation, Aachen University of Technology.

Meyer, B. (1994) Reusable-Software - The base objectoriented component libraries. Prentice-Hall.

Pree, W. (1995) Design Patterns for Object-Oriented Software Development. Addison-Wesley ACM-Press.

Pritschow, G. (1991) Leit- und Steuerungstechnik in flexiblen Produktionsanlagen. Carl-Hanser Verlag, München.

Schmid, H.A. (1994) Kundenspezifische Software zur Fertigungsautomatisierung durch Wiederverwendung eines objekt-orientierten Baukastens. CIM Management 6/94 pp.50-54

Rumbaugh, J. et.al. (1991) Object-Oriented Modelling and Design. Prentice Hall.

Stroustrup, B. (1992) The C++ programming language. Addison-Wesley.

Weck, M. (1995) Werkzeugmaschinen-Fertigungssysteme Band 3.2, Automatisierung und Stuerungstechnik. VDI-Verlag, Düsseldorf

Wirfs-Brock, R. et.al. (1990) Designing Object-Orinted Software. Prentice Hall.

Zipper, B. (1994) Das integrierte Betriebsmittelwesen - Baustein einer flexiblen Fertigung. Forschungsberichte iwb, Springer-Verlag, Berlin.

7 BIOGRAPHY

The Authors:
Prof. Dr.-Ing. Dr.-Ing. E. h. Manfred Weck, born in 1937; head of the Chair of Machine Tools since 1973 and member of the directorate of the laboratory for Machine Tools and Production Engineering (WZL) of the Rheinisch Westfälische Technische Hochschule Aachen (RWTH).
Dipl.-Ing. Jörg Friedrich, born 1964, studied Mechanical Engineering at the RWTH Aachen, engaged in scientific research at the WZL since 1991, Chair of Machine Tools, Controls of Manufacturing Systems group.
Dipl.-Ing. Thomas Koch, born 1963, studied Mechanical Engineering at the RWTH Aachen, engaged in scientific research at the WZL since 1989, Chair of Machine Tools, Controls of Manufacturing Systems group.
Dipl.-Ing. René Langen, born 1968 studied Mechanical Engineering at the RWTH Aachen and the Center for Advanced Manufacturing, Clemson S.C., USA, engaged in scientific research at the WZL since 1994, Chair of Machine Tools, Controls of Manufacturing Systems group.