A New Architecture for Flexible Shop Control Systems*

Gerrit Teunis, **IFW**, *Paulo Leitão* **CCP**, *Michael Madden*, **MCS**

Abstract:

In the last two decades, there has been a marked trend in manufacturing away from function-centered production organisations towards product-based manufacturing shops and smaller organisational units. These new organisational structures reduce management complexity and facilitate greater human involvement and empowerment. Modern manufacturing facilities must be flexible, to allow for rapid reconfiguring of resources as dictated by variable demands. Today's tools for production planning and scheduling are not flexible enough, most of them being based on a certain production structure. A newly-developed kernel/shell technology aims at a software architecture for shop control not being based on any assumption of the production structure. It should be possible to support almost any structure, organisation and scheduling algorithm. A central kernel acts as a service provider for a broad array of shell modules for scheduling, interfacing with legacy systems, database I/O, user interaction, SCADA interfacing, etc. This technology for shop control provides a high degree of customisation, flexibility and extendibility.

Keywords: *Production Management, Shop Control, Scheduling, Decentralised Systems*

1. Introduction

Since the 1980s, the manufacturing industry has been shifting from functional production arrangements towards product-based manufacturing shops and smaller organisational units (e.g. lean management and group technology) [1,2]. These small, autonomous production shops and cells need visual, user friendly scheduling and control systems, which can be customised and integrated with internal and external supplier network. At the same time, production facilities are being restructured more and more frequently. This is expected to become an almost continuous process, aiming at structuring the production according to the current order situation at any time.

Since current tools for production planning and scheduling are mostly based on a certain production structure and centralised planning, they are not flexible enough to support frequent organisational changes [3]. Usually, a central system for production planning and control (PPC) is used for rough level planning (production program planning). Detailed planning (scheduling, resource allocation) and control is done by a shop control system. These tasks are more and more decentralised and shifted to the shop floor level. Central planning system will become less important and scheduling and control decisions will be made independently in autonomous production units (Fig. 1).

A variety of shop floor control applications written in different languages has to be supported. To make communication between applications easy, permitting the rapid installation of new software, a common architecture solution is required.

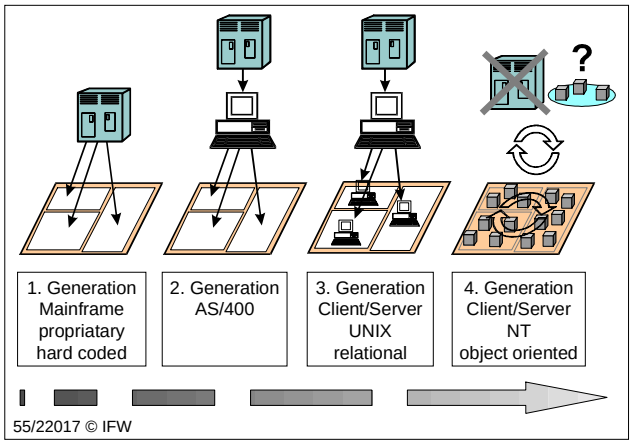

Fig. 1: Developments in Shop Control and Planning

2. Shop Control Systems in Manufacturing

The Shop Control systems solutions must meet the following objectives in order to fill the Shop Control market aims:

- *It must be user-friendly and visually-orientated*
  A software product to be used at the shop floor level, or very close to it, must allow almost intuitive interaction.
- *It must be highly flexible, so that it may be customised easily for different types of production structure*
  Shop control software has to adapt to the organisation of the manufacturing environment. With the goal of continuous optimisation, this environment is restructured and reorganised frequently. It should be possible to update the shop control tools easily in order to support such changes. For example, it should be possible to change the systems scheduling module or database interface module without shutting down

* The investigation that is reported in this paper is supported by the European Union, Innovation Programme IN 20563D, "A Modular Shop Control Toolkit for Flexible Manufacturing".

the software system or in any other way disturbing the operation of other modules.

- *It must be possible to integrate the shop floor control system with existing scheduling systems and upstream production planning and control systems*
  Stand-alone solutions are not acceptable any more in modern manufacturing industry.
- *It must allow the interfaces to Other Systems*
  The integration of existing data and systems becomes more and more important. Thus, shop control systems provide interfaces for some important systems like SAP R/2 and R/3, Baan, IBM-Mapics, Copics and other standard interfaces to PPC/MRP or SCADA systems. These interfaces are used to get shop order data.
- *It must facilitate co-operation and communication between different manufacturing cells within a manufacturing plant*
  Flattening of organisational structures creates the need for horizontal communication and co-operation. The vertical command hierarchy is replaced by direct interaction on the shop floor level. For example, synchronisation between assembly and machining shall be based on the communication facilities of the corresponding local shop control tools.
- *It must allow communication between suppliers, manufacturers and customers, thereby permitting a greater degree of integration along the external supply chain*
  As an extension of the above-mentioned interaction between for example assembly and machining, these direct, horizontal co-operation and communication links more and more often expand beyond the limits of the physical enterprise into what is termed the extended enterprise.
- *It must support customer service*
  The installation of a complex shop control system is a time and cost expensive task that requires a thorough support by both, the system provider and the user. Most software companies offer a comprehensive support for installation and maintenance of their systems. Beginning from an analysis of the users production, consulting, training, work shops, project teams, hot line and remote support.

Existing shop control systems have reached a mature state in terms of performance and functionality. Anyhow, they are not flexible enough to support changes in the organisation structure or in the data base. Once a customer specific design of the software and the database is defined it is difficult to change it again. Although a lot of systems support client-server applications it is not possible to distribute single modules of the system on different computers. It is only possible to run instances of the whole system on different machines. The modular design of these systems is limited to one single user specific installation. They are still based on a monolithic structure and offer flexibility only at the beginning. Afterwards, the systems can be upgraded or some functions can be added. A complete restructuring of the whole system is not possible. To add customer specific functions the standard system does not provide a considerable and expensive effort is necessary. Furthermore, the integration in other systems is only possible via existing interfaces to some of the big PPC systems like SAP or Baan. In praxis, most SMEs don't have these big systems but other low or mid range systems without any interface to existing shop control systems. Finally, there is only a rudimentary support of decentralised structures. Most Systems are oriented to a traditional central shop control without any support of modern organisational structures like group technology.

A new kernel/shell technology has been developed as the base for a Modular Shop Control Toolkit (MoscoT) that supports almost any structure, organisation and scheduling algorithm. This technology allows the development of user-friendly, flexible and highly customised shop control systems that meet the requirements of modern manufacturing.

3. The Moscot System

In spite of existing high performing shop control systems on the market MoscoT offers a number of unique features previously not realised in shop control. Rather than a new software product among other systems MoscoT is an innovative approach to a new architecture for shop control. It is designed to be low-cost, intuitive to use, customisable, adaptable, and well suited to modern manufacturing organisational structures. Since manufacturing structures develop rapidly a manufacturer cannot know how his organisation will look like in several years. Thus, he also cannot foresee the requirements of his shop control system. For that reason MoscoT does not assume any fix organisational structure and can easily adapt to every change.
The competitiveness of a manufacturer, and therefore the potential to lead the market, comprises three major factors: a) product know-how, b) marketing know-how and c) business and manufacturing processes know-how. To be successful the manufacturer has to offer a good product (technology leader, innovative, unique, etc.), he has to show good marketing skills (customer relationship, pre- and after-sales support, market research, advertising) and he has to optimise the processes, both business and manufacturing as the basis for quick reactions, reliable output and cost effectiveness.
Although not a business re-engineering tool, MoscoT is intended to support this third area. Too often, re-engineering and restructuring efforts are slowed down by inflexible software solutions in the area of planning, scheduling and control [4]. It is impossible to predict the nature of the changes in manufacturing organisation and production processes. Thus, it is impossible to design a software product which delivers static re-configuration options for today's and tomorrow's customisation needs.

So, the intention of MoscoT is to establish a new architecture for shop control systems that meets the requirements of modern manufacturing. The innovative approach of MoscoT basically consists in five areas:

- MoscoT is completely based on a modular design. This is realised by a kernel/shell architecture that concentrates all general functions and data in a central

kernel. User specific functions are implemented in a comprehensive set of shell modules.

- MoscoT provides an open, fully documented programming interface that allows third parties to develop own shells and to integrate existing systems.
- MoscoT is especially designed to support decentralised structures. Therefore, a special shell module co-ordinates the co-operation and communication of different shop control systems.
- MoscoT offers a flexible, user-friendly shop control system that allows high customisation, low costs for implementation and maintenance and an easy reconfiguration of the system.
- MoscoT benefit from the latest hardware and software technologies available as industrial standards on the market. MoscoT runs on PCs under Windows NT which allows an effective use of network technologies. With MoscoT it is possible to run a shop control system of different machines distributed on different sites, at the limit all over the world.

The description of each innovative area of the MoscoT system will be done in the following points.

3.1 A Kernel/Shell Architecture for Shop Control Systems

The MoscoT approach is to offer a new concept for inexpensive, user specific software which can be reprogrammed with relatively low effort. The system is based on a central *Kernel* and a set of *Shells* which operate together to deliver the required functionality. The fundamental idea of the architecture is that several functions of shop control are independent from the type and size of production the MoscoT system concentrates all common functions of shop control in a central kernel. All customer specific functions are implemented in a set of shell modules.

The kernel is a collection of basic data structures and services which are common to the majority of applications. The kernel itself can not be used for any shop control tasks as it lacks a user interface and all customer specifics. On top of the kernel reside the shell modules which provide the functionality commonly known for shop control tools, e.g. Gantt-chart like GUI, scheduling algorithms, maintenance functions. Each MoscoT shell is independent of all others. All shells operate through a common interface and have access to each other's functionality via the MoscoT kernel.

The modular design of the system guarantees a maximum degree of flexibility and adaptability. New modules can be connected to the system, disconnected from the system and reconfigured without the system needing to be restarted or in any way interrupted. The user can build up his individual system from existing shell modules or new, user specific shells can be developed by third parties or the user itself.

3.2 An Operating System for Shop Control

Since the kernel is the co-ordinating module of the system it includes a central data repository, a library of functions which other modules use, and administrative functionality for supporting the interaction between modules.

The kernel acts as an operating system for shop control. The operating system of a PC for instance provides general functions that allow the user to run his own applications or to develop new ones. Therefore he can use existing functions for example a dialog box to open a file and he does not have to rewrite them. When these basic functions are updated with a new release of the operating system the user can immediately benefit from without needing to recompile his application.

According to these principles MoscoT provides a new 'operating system' for shop control. All production type independent data and functions are implemented in the central kernel. All user specific functions and data are implemented in the shells, the shop control application programs.

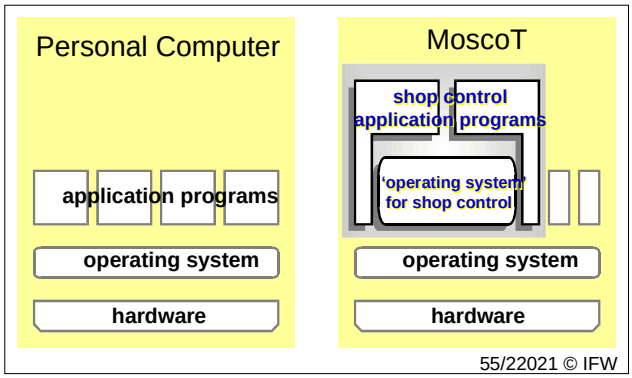

Fig. 2: MoscoT as an operating system for shop control

Shop floor data is stored within the kernel using a hierarchical class structure. This hierarchical class is structured in five function areas that were identified to be common to almost any type of production:

- Bills of material,
- Process plans,
- Resources,
- Orders and
- Interfaces that support the modelling of common manufacturing data entities.

The shop control functionality is independent from the kernel. Since shell modules can easily be exchanged the functionality of one system can be completely different from another system the is based on the same kernel.

3.3 A Shop Control Application Programming Interface

The kernel has an open, fully-documented programming interface. Any software developer or a manufacturing company's IT department can develop new special-purpose shells which interact seamlessly with the Kernel and existing shells.

Shell modules can access all of the data and functionality of the Kernel through a functional interface. The interface which the partners proposed and developed is called SCAPI, an acronym for *Shop Control Application Programming Interface*. An Application Programming Interface, or API, is a generic term used to describe a library of functions which applications use to gain access to operating system services. For example, the Windows API supports creation of windows and menus, control of

the pointer, file access and so on. In an analogous manner, SCAPI provides functions to get/set Kernel data, access to methods implemented in the Kernel, and functionality to support message-passing and event notification for the Kernel and shells.
Fig. 3 shows the relationship between the kernel, SCAPI and the shells. SCAPI aims to provide a comprehensive set of methods and data access functions to objects which reside in the MoscoT Kernel. These may be used by MoscoT shell applications, which work together to provide the functionality of the MoscoT shop control system.

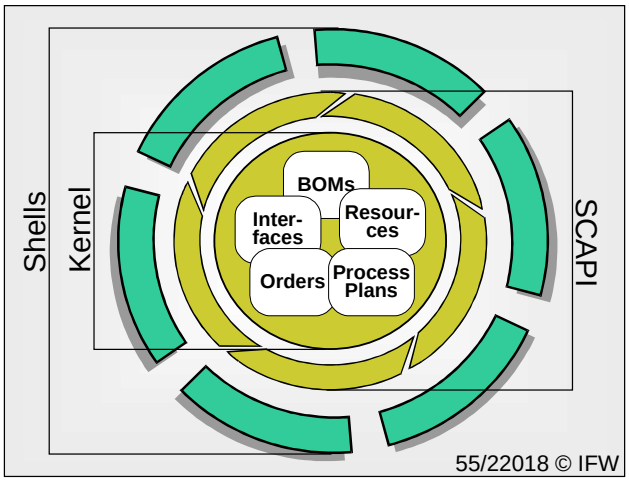

Fig. 3: Relationship Between the Kernel, SCAPI and Shell

The modular design of a shop control system does not necessarily require a standardised interface for the modules. Several existing systems provide modules of their system for an individual customer specific set-up of the system. But SCAPI does not only support the modular design of MoscoT coordinating the activities between the shells and the kernel. SCAPI is an open, fully documented programming interface that allows any software developer to develop shells which interact seamlessly with the MoscoT system. Thus, SCAPI provides a simple and effective mechanism for real-time data interchange for shop control software. Software ranging from large upstream production planning and control systems to small add-on applications and university-developed code can benefit from compatibility with SCAPI.

3.4 New Software Technologies for Shop Control

System Platform

The MoscoT system is based on inexpensive and widely used technology: PCs and PC operating system compatible with Windows. The envisaged, fast-growing market requires an inexpensive, easy to handle, and flexible tool for shop control.
PC hardware technology is developing very quickly, to the point where the traditional performance gap between workstations and PCs has recently become an overlap. Thus, PC hardware nowadays offers workstation performance for relatively low costs. Another major advantage of PC hardware in view of the envisaged decentralised shop control application is the widespread PC experience of users (thus less training effort is required) and the parallel use of standard office applications or the direct link from shop control to these applications. For example, a company's schedule database may be linked to a spreadsheet for monitoring and evaluation of characteristic values for workshop performance.
So, powerful PCs are inexpensive and widely available, as is communications technology such as modems and Internet/intranet software. Newer PC operating systems, like Windows NT, are not based on an underlying DOS and offer UNIX-like features combined with low set-up and administration effort.
The use of PCs hardware and Windows based platform, allow to MoscoT system a good market penetration, an inexpensive and compatible platform, a wide range of market leading development tools available and a network and Internet support.

OLE Automation and DCOM

Distributed, decentralised shop-control and higher autonomy and co-operation require an appropriate communication infrastructure. If in particular a shop-control system is to allow for co-operation along the external supply chain, communication becomes a vital issue.
MoscoT uses a component technology with a client/server model for data exchange. This model is based on widespread industry standards. The Component Object Model (COM) is a technology that allows to use components of an application within another. A typical example is a spreadsheet that is linked to a text document. The spreadsheet can be manipulated directly in the text document without needing to start the spreadsheet application. This is frequently used for compound documents but has a more general importance. The technologies that use COM are called Object Linking and Embedding (OLE). In MoscoT OLE provides objects of a module to another module. For example, when a shell module gets information about a certain shop order it calls the kernel that acts as an OLE server providing the required object. This method allows to provide all data and functions of the kernel to the shells.
SCAPI is implemented as a fixed number of OLE servers which reside within the Kernel. Each OLE server serves a particular class of Kernel objects. For example, there is an OLE server for process plans and one for manufacturing orders. Shells connect to these OLE servers to gain access to the Kernel. Some of the key features of the data exchange between the kernel and the shells are as follows:

- Kernel objects can access each other directly, they do not use SCAPI.
- SCAPI is comprised of exposed methods of OLE servers giving access to Kernel objects.
- SCAPI provides an interface which is visible to shells.
- Shells cannot access internal Kernel objects directly, they must go via SCAPI.
- Shells cannot access each other directly, they must go via SCAPI in the Kernel.

The Kernel contains methods which implement services that are needed by the majority of shells. These methods

are made available to the Kernel by being exposed via OLE. The Kernel is an application having objects which in turn have methods and data, and the shells are separate applications which may create and manipulate the Kernel's objects. In OLE terminology, the Kernel generally operates as an *OLE Server* and the shells generally operate as *OLE Clients*.
However, the Kernel will not always be operating passively. As well as making methods available for shells to access, the Kernel must be able to access shell methods, particularly in order to activate the shell. In such situations, shells operate as OLE servers and the Kernel operates as an OLE client. It is perfectly acceptable for the same program to operate as both an OLE client and server.
The primary advantage of using OLE for SCAPI is that is a language-independent technology. Therefore, it is possible to implement a shell in one of a variety of languages. Because it is language-independent, shells may be (and are) developed using a variety of programming languages supporting OLE such as Visual C++, Visual Basic and Borland C.
Fig. 4 shows an example how a shell accesses a kernel object using SCAPI and OLE. In this example the Graphical User Interface loads all corresponding work order data using SCAPI's OLE function *GetWOData()*.
The GUI shell represents the schedule as a Gantt chart with the color and pattern of the bars corresponding to the type and status of the orders. If the user starts for instance a work order the color of the bar changes. The GUI shell actualises the corresponding kernel object using the SCAPI function *SetWOData()*.

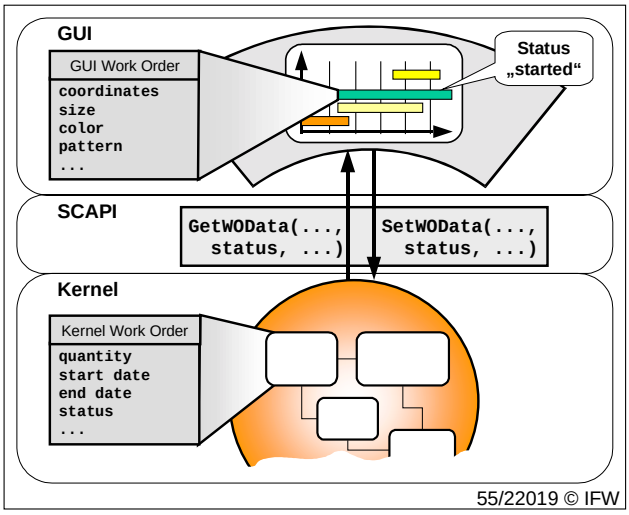

Fig. 4: Access to a kernel object using SCAPI

The use of COM is limited to software components on one single computer. Distributed COM (DCOM) extends the COM model to support communication among objects on different computers on a LAN, a WAN or even besides the public internet, allowing the implementation of distributed applications. DCOM uses Remote Procedure Calls (RPC) to call a server. The client can call a server on another computer as if it was implemented on the same machine. DCOM allows the development of location independent, scaleable and protocol and language independent software components. This makes it possible to distribute modules of a shop control system on different computers.
For example, the kernel can be implemented on a powerful computer in an office. A less performance sensitive shell like a data collection shell for reports and data feed-back can run on a simple PC in the shop floor. This way MoscoT becomes more scaleable and customisable. Moreover, a manager could even get the status of an order running a Graphical User Interface on his laptop to access a shop floor kernel directly via the Internet.

3.5 Decentralised Shop Control

An important feature of MoscoT is the support of decentralised structures. When planning and control is decentralised and shifted to semi-autonomous production units co-operation and communication between these units becomes more and more important. This has to be supported by effective tools for message exchange to co-ordinate the interaction of several systems.
With MoscoT it is possible to connect several shop control systems in one manufacturing site of a company, in different sites or even at suppliers or customers. A kernel-link module is used for data exchange between different shop control systems. Shop control areas can be defined according to the specific needs of a manufacturer. For instance a shop control area can be defined in manufacturing another one in assembly. The user defines the degree of decentralisation of shop control that suits the company's needs. No central supervisory system is needed to co-ordinate all shop control systems. Decisions can be made locally which allows a high degree of autonomy.
Although acting independently schedule changes in one shop control area may affect other areas. Nevertheless, all shop control areas have to cooperate and communicate very closely. Any change of the schedule in one area may affect other areas (Fig. 5). The kernel-link shell forwards each changes of the kernel data to other connected kernels. If for instance on order in a shop control area A is delayed other areas have to be notified. If this delay does not affect area B only the kernel is updated. If a shop control area C is affected by the delay its GUI shell has to be updated, too.

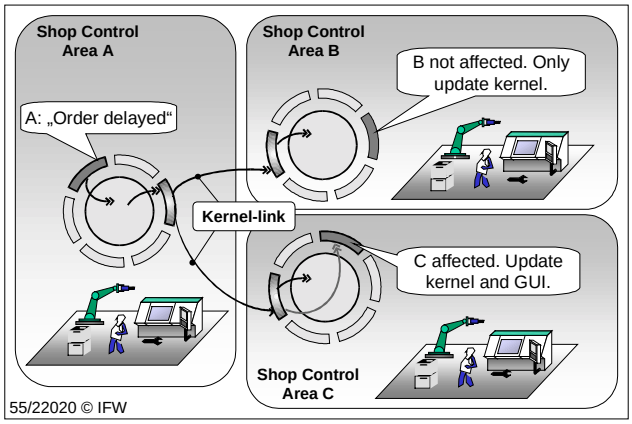

Fig. 5: Connection of Different Shop Control Areas

Within one site the users (foremen, schedulers, planners) require a full access to all MoscoT data and functions,

regardless of the MoscoT control area boundaries. Therefore all MoscoT systems share the same data, except the identification of the system to itself. Every MoscoT system must know to which shop control area it belongs, e.g. MoscoT system A 'knows' that it controls shop control area A. But the scheduling data (the kernel object instances) is not limited to its own control area. Every MoscoT system within one site shares the full set of data with all other MoscoT systems. Thus the data is duplicated, as every kernel contains the same data. This mirroring mechanism also improves the security of the system and is a good safe-guard against data losses.
The Kernel-link-shell can be implemented on different technology platforms according to the specific needs of the user. The purpose of the Kernel-link-shell is to support the co-operation of different shop control systems. Therefore all implemented kernels are synchronised and share the same data. Often, it is not necessary or even required that not all connected kernels share the same data or have access to all data. This may be the case with external MoscoT systems or systems in different shop floors. For this purpose the Kernel-link-shell provides a filtering mechanism where the user can define what kind of data will be synchronised and which kernels will be synchronised.
The MS-Mail based version can be used for the communication between MoscoT systems that are not located in the same shop floor. For example a supplier's MoscoT system is not supposed to be synchronised with the customer's MoscoT system. The only purpose of a connection of both systems is a weekly message about the status of some orders. In this case a Kernel-link shell sends an E-mail to the other kernel.

4. Realization of the Kernel/Shell Architecture

To implement the kernel/shell technology a prototype was developed in order to verify the concept of this architecture and to identify strengths and weaknesses of the used technologies [5]. The focus of this development was the kernel whereas the shells basically serve as test applications for the kernel.
The current prototype consists of the kernel and a set of 8 shell modules (Fig. 6). A Database Interface (DBI) acts as an interface between a local database and the kernel. It loads order independent data to the kernel. Since MoscoT does not require a certain database structure almost any database can be used. The DBI shell can also be used to make backups of the kernel data.
For the three end users involved in the prototype development specific PPC interfaces were developed to integrate their upstream PPC/MRP systems running under AS400/UNIX. The PPC interfaces can introduce and request data from these systems.
For graphical representation of order data a Graphical User Interface is used to display schedule data. Therefore, a third party OLE software component was integrated. The GUI shell also allows the manipulation of order data (start and end dates, statuses, etc.) and the creation of new orders.
The detailed planning can be optimised using an automatic scheduler. It automatically generates a schedule which can be modified or accepted by the user. He can select a combination of the most common priority rules (First-In-First-Out, Shortest Operation Time, Least Slack, etc.) in order to find an optimal schedule iteratively.
Start and completion of orders can be reported with a SCADA module for shop floor connection. It allows the shop floor operator to monitor and change the status of an order. Like other modules this shell can be implemented on a different computer for instance on the shop floor.
Another module is the kernel-link shell that serves for connecting different shop control systems. The use of this shell is described below.

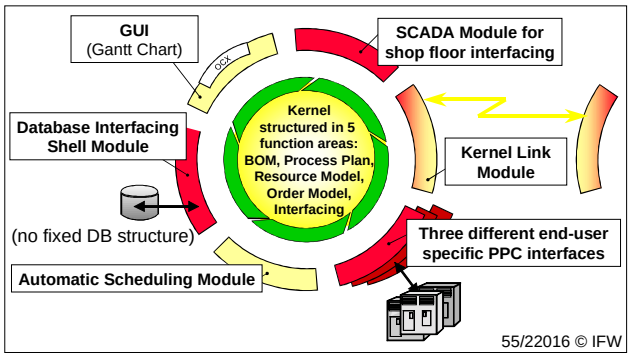

Fig. 6: Overview of the MoscoT system

5. The MoscoT Project

The objective of the EU sponsored project MoscoT („A Modular Shop Control Toolkit for Flexible Manufacturing", Innovation Program IN 20563D) is to develop a toolkit, using a kernel/shell architecture, which will match the requirements of modern manufacturing industry. The MoscoT toolkit will consist of a central kernel and a comprehensive set of shell modules that can be selected and combined via the kernel to produce a shop control system which is tailored to each manufacturer's specific needs.
During the project an existing prototype implementation of the kernel/shell architecture will be developed to a marketable shop control system. Since the objective of the current prototype was to validate the kernel/shell architecture most of the modules need considerable redesign.
MoscoT will be developed in close co-operation with the European industry. Three manufacturing companies are involved in the consortium. They will implement a MoscoT system as soon the system reaches a sufficiently mature state. At the end of the project they will have a functional highly customised shop control system to improve communication and co-operation within the company and along the supplier chain.
Additional end users can implement prototypes of MoscoT in order to evaluate and improve the software and to develop a shop control system according to their individual requirements.
Furthermore, it is planned to establish SCAPI as a de facto industrial standard. Therefore, a licensee network for the kernel technology and the kernel product, including a strategy to cater for supplying different collections of toolkit components to different users according to their requirements.

6. Conclusions

In order to meet the requirement of modern manufacturing with frequently changing, decentralised structures a new software architecture for reactive scheduling and shop floor control has been developed. It allows the configuration of flexible, low-cost and user friendly shop control systems that can easily be reconfigured and integrated in existing systems. The system is based on a kernel/shell technology where the kernel acts as an operating system for shop control. The shell modules provide specific shop control functionality according to the company's need. Modules can be replaced or added during run-time which allows an easy re-configuration of the system.

An open fully documented Shop Control Application Programming Interface allows the development of modules by third parties or the end user itself. An implementation of several shop control systems makes possible decentralised planning and control within one manufacturing site, between different sites or along the supplier chain. This technology is a promising approach to offer a previously unreached degree of flexibility and adaptability.

References

1 *Wiendahl, H.-P.; Garlichs, R.:* Decentral Production Scheduling of Assembly Systems with Genetic Algorithm. Annals of the CIRP 43 (1994) 1.

2 *Fraser, J.:* Scheduling: A Question of Balance. American Production and Inventory Control Society Journal, 6 (1996) 3.

3 *Aupperle, G.; Burr, G.:* Der Leitstand: Werkzeug für dezentrale Organisationsformen? Industrie Management 13 (1997) Sonderheft PPS Management.

4 *Tönshoff, H.K.; Winkler M.:* Shop Control for Holonic Manufacturing Systems. CIRP Proceedings - Manufacturing Systems, 25 (1996) 3.

5 *Madden, M.; Uusitalo, M.; Winkler, M.*: Towards Distributed Computing and Decentralisation in Shop Control. Proceedings of the Manufacturing System Design '97 conference. Magdeburg, Germany. 15-16 May, 1997. Paper 37.

6 *PASO Consortium:* Final Report, July 1997.