

Object-Orientation in Planning and Control of Decentralised Production Systems

Fraunhofer
Institute for
Industrial
Engineering
(IAO)
Stuttgart,
Germany
Fidia S.p.A.
San Mauro
Torinese (TO),
Italy

Thomas Schlegel
Wolfgang Beinhauer
Fabrizio Meo

Targeting the need for intelligent, integrated models and components in order to create rapidly reconfigurable manufacturing systems, this paper presents an approach for fundamentally integrating the object-oriented paradigm into production environments. The goal is to operate machines and adjacent systems in a more flexible and fail safe way by applying a new approach that deploys a meta model which makes machines capable of describing themselves. Communication with other machines or computing systems is established in a decentralized and message-based manner. By applying basic concepts of object-orientation to all artifacts of a manufacturing plant, the production environment and machine states become interpretable, which allows for new features in maintenance and production control.

Keywords: object orientation, object-oriented modelling, production control, semantic messaging, service oriented architectures, decentralised production.

1 Introduction

An increased level of automation and rapid responsiveness to changing market conditions is emerging as a key factor for successful manufacturing enter-

prises. The driving force behind the automation and maintenance is the need for shorter development cycles and for a broadening range of products.

Markets still tend to demand more and more flexibility in production triggered by a trend towards mass customisation [I*PROMS POM 2005] leading to a demand for rapid reconfigurability of machines, cells, production lines and even whole factories. Together with the necessity of short rule cycles and integrated data this leads to a growing convergence on the system and data integration level, which is indicated by developments such as Manufacturing Executions Systems (MES) [Kletti 2006] that integrate different aspects of production into one central system.

The migration process from individual production control to complex centralized systems can be regarded analogously to the partial migration from the "code&fix" approach in software engineering to heavy-weight software development process models like Rational Unified Process (RUP). Likewise, we can expect the introduction of central MES systems in production to have a positive effect on the manufacturing quality. Thus, the next quantum leap in software engineering is the evolution from centralized heavy-weight procedures to lightweight, more decentralized development processes. MES being client-server systems are benefiting from systems with a high level of (proprietary) integration. On the other hand, such kind of systems tends to be large monolithic complexes that introduce a bottleneck and single point of failure problematic. Reliability is often improved by redundant core modules which still does not solve the problem of network ruptures.

While in Software Engineering light-weight methods have evolved and distributed systems

have partially been decentralized, production planning and control systems still lack concepts for a common/integrated data and semantic model with a decentralized platform infrastructure capable of dynamically embedding and accessing new system and model components on run time. In the following, we describe how this kind of technology can be introduced to manufacturing systems.

The goal of this technology is to achieve type-based recombination of products and processes. Former concepts used manual reconfiguration and revision of software, later approaches allowed for switching of preprogrammed processes. Dynamic reconfiguration of processes is still an issue leading to many new approaches. The approach presented here targets even implicit reconfiguration based on demand classification in an object-oriented production environment.

This can only be achieved with a smart connected platform containing semantic models which are interpreted, modified and executed on run time and are not contained within software applications but dynamic executable models.

From the point of view of computer science, there is a tendency towards decentralized but model-based systems targeting easier development, integration, compatibility and change management for systems in the field.

Roadmaps [I*PROMS POM 2005] show that key enabling features for sustaining under current and expected future market conditions clearly go in the direction of flexibility and model-based systems to cope change and complexity.

So far, object orientation (OO) has been used mainly for describing the paradigm of structuring and developing software programs, for example in the

domain of embedded systems. More recently, OO has been applied to defining and designing software components and larger systems to be integrated in automation [Vyatkin et al. 2005].

This approach is not only targeting development technology but a model-based rapid reconfiguration executed at run time with dynamic object-oriented models adapted just in time within the frame of a common meta-model.

The graphical representation of the Unified Modelling Language (UML, [OMG 2005d]) notation is used here for describing inheritance, aggregation and other modelling concepts available in UML.

2. Concepts of Object-Orientation in Production Planning and Control

Most approaches to production control and enterprise modelling in the manufacturing industry suffer from the lack of completeness of their expressivity. In particular, the lack of vertical enterprise integration is one of the major shortcomings of conventional systems and modelling techniques, respectively.

In order to support agile business transformation and quick market responsiveness, a consistent picture of both the production level as well as of the organizational level has to be taken. The same way as key business indicators (KPIs) are aggregated and visualized by business intelligence tools, sensor data collected during production needs to be collected and interpreted in order to provide valuable knowledge about machine abrasion and maintenance schedules. However, both levels cannot be separated from each other since e. g. machine failures might have string impact on the KPIs defined in the business level. Likewise, business decisions might affect key parameters on

the production controlling level such as schedule times.

Moreover, common approaches to enterprise modelling are widely lacking of horizontal integration capability. Tools for production planning are mostly apart from tools for machine control. However, rapid market responsiveness requires a quick implementation of newly defined production processes or even automatic self-adaptation of processes. The same yields for the rapid formation of partnerships and the implementation of cross-enterprise business processes.

Hence, what is required for the realization of the vision of the real time enterprise is an holistic approach that covers both the organizational level and the production level, and that is suitable not only for production planning at design time but that supports direct implementation of processes in terms of a run time environment. This entails difficult requirements for any modelling language or notation used. Whereas the requirement to support executable process descriptions pushes the modelling towards a rather stiff formalized and machine-interpretable format, the desired expressivity of production steps needs a much higher degree of freedom and extensibility.

Various approaches have been undertaken in order to enable a vertically or horizontally integrated representation of a manufacturing company. The application of object-oriented modelling was already proposed for the creation of executable models. However, its expressivity was limited to design-time modelling only [Müller 2007] – behaviour could not be specified and changed on run-time. Alternatively, a semantic modelling language such as OWL [Patel-Schneider et al. 2004] possesses sufficient expressivity to cope with production techniques. With Pellet [Sirin et al. 2007], a quite performing run time environment is given. However, the graphical notation of OWL ontologies [Gomez-Perez et al. 2004] is not intuitive and least process-oriented, which prevents it from being suitable for our purposes. In the sequel, a new hybrid approach is presented that combines the advantages of linearised process descriptions with the high expressivity and openness of OWL. It is based on the paradigm of object-orientation.

The concepts explained here match at the implementation level for example with W3C standardized technologies such as eXtensible Markup Language (XML), XML Metadata Interchange (XMI) [Grose et al.

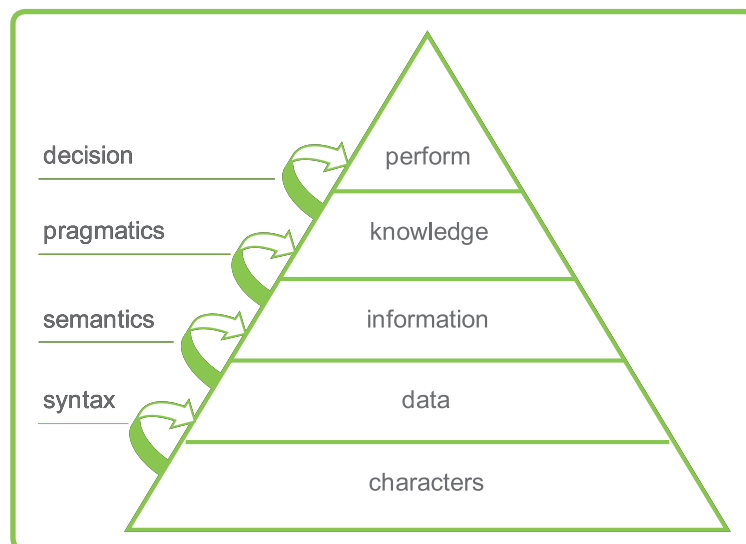


Figure 1:
Extended Knowledge Pyramid on the Basis of [Müller 2007; Wolf et al. 1999]

2002], Simple Object Access Protocol (SOAP), Resource Description Framework (RDF), Web Ontology Language (OWL) etc., which can be used for model exchange and implementation purposes. Besides these formats and interpretations, these concepts may also be implemented using more flexible and efficient structures internally. The mapping to and use in production environments of the modelling concepts explained below represents the deciding step.

3. Object Orientation in Production

The object-oriented paradigm is very close to the mental models of humans compared to approaches like functional models. [Bergin 2006] Objects and their interactions are more close to real life as well as classification is a regular task of humans in order to understand and facilitate perceptions made. Therefore, the paradigm of object-oriented production is used on two layers of abstraction: using object-oriented models to describe elements and communicate in a Production environment is supplemented by machines and other parts/systems acting as objects themselves.

3.1 Component-Relation Models

The first step for achieving a general model is the definition of a general model artifact, which we call component. Every item within the system – virtual like a type or real as a machine tool can be defined as a component, serving as a basis for further definitions.

Associated with components are capabilities. The most important one at this stage is identity. Giving each item in a production environment a unique identity regardless of its abstraction layer lays the ground for a powerful and distributable model advanc-

ing from the stage of Manufacturing Execution Systems (MES), which are trying to integrate different layers, aspects and data sources within one frame, to an integrative, decentralized system composed of software components connected by a messaging framework.

Components can be connected with different types of relations. The most important and complex one is inheritance – especially multiple inheritance. Also very important is the (hierarchic) aggregation of components. A proper classification concept for relations is needed to ensure type-specific reactions of the system and the ability of verification based on rules and types.

3.2 Aggregations and Composites

Aggregation and composition are the main concepts for building complex structures, which acquire identity themselves as aggregates or composites.

Ordered aggregates allow for concatenation of components $c_1..c_n$ within an Aggregate component a_1 to create causal structures with one or more components $c_m..c_n$ following other components $c_1..c_{m-1}$ ($1 < m \leq n$).

Including time into the concept, causal sequence (ordinal scale) can be valorized to a temporal meaning (ratio scale) describing already computable processes. These time data are associated with components (mainly process steps) rather than relations.

3.3 Inheritance

Inheritance is the most powerful concept of object orientation. Even with single inheritance it is already possible to build complex and powerful classification structures from atomic types, for example classifying artifacts, events, machines and tools in a production system.

Using aggregations, complex components can be structured and by inheritance spread over a wide range of sub types or variants. Inheritance makes it possible to describe features of a general type of tool or machine just once and then use specialization to derive variants from such a generic machine type inheriting all associated information from the base type.

The same applies for work flows and processes in production. A method has been developed at Fraunhofer IAO to specialize work flow structures by inheritance, which allows for adapting a whole group of work flow variants by just adding or changing a step in their general process type. Inheritance will then transitively conclude or automatically propagate the new process step down to all variants derived from the work flow changed.

This can add semantic depth to formerly limited or semantically flat models such as Computer Aided Business Process Management (CA-BPM) [Frankel 2004], Enterprise Resource Planning (ERP), Computer Aided Manufacturing (CAM), Computer Integrated Manufacturing (CIM), Production Planning Systems (PPS) and many other IT-based approaches in production systems. A common meta-model and a decentralized system will also ease integration with remaining external systems such as ERP or supply chain management solutions.

3.4 Instantiation and Instances

Most approaches that are dealing with Object-Oriented Programming and software development focus on the class layer, which defines types, their structure and their relations to other types. Instantiation on the other hand is more important on run time than when developing systems.

While classifications serve as a basis for drawing conclusions on

run time, instances can provide items in a production system with an identity while preserving their typological information for the systems to be interpreted.

Using a holistic model, each item in a production system is described as an instance of an item type. This leads automatically to each item having an identity, state, type and descriptive information. From a knowledge management point of view, for each item (instance) its specific information like quality history and RFID value can be provided and accessed. Following the instantiation path, it is as well possible to access type information and characteristics like hardness of this kind of item, handling instructions, problem record etc. which apply to all items of this kind. Especially for quality management issues, it is necessary to relate both, class information and instance information. Relating a module instance to its charge – both on instance layer – provides all special information about the charge together with the ability of logging quality data based on and related to these instances.

Although instances form a strong mechanism for creating items and work flow instances ("projects") on run time, difficulties arise when class structures are changed while active instances exist. For example a production work flow WA for producing a module type A could be changed while a125 – an instance of A – is currently assembled. Changing the work flow could cause unsolvable inconsistencies in the production of a125 as part of it has already been produced using the old process, while now the changed version should be applied.

3.5 Common Meta-Model

Stand alone systems can directly profit from an implementation of the above model concepts. To achieve the same in a het-

erogeneous and decentralized environment, it is necessary to define common standards for information exchange within the production system as it has been done for Enterprise Application Integration, Common Object Request Broker Architecture (CORBA) [OMG 2004b], SOAP, OWL, XMI, Ethernet and other transmission and data exchange standards. The Enterprise Service Bus (ESB) [Keen et al. 2005] is one approach for operational application integration but lacks a common superstructure.

Defining a common meta-model structure that specifies and uses the above concepts – also cyclic for its own definition – allows for understanding new partial models and exchanging semantic information between multiple peers working on different partial models under the same basic meta-model. An existing approach for such an object-oriented meta-model is the Meta Object Facility (MOF, [OMG 2006e]) that has been defined by the Object Management Group (OMG). As basis for the Unified Modelling Language (UML) 2.0 [OMG 2005c; OMG 2005d] MOF-based models share a common meta-model that allows for interpreting their structures by a MOF-compatible system. Using MOF-based and similar models, it is always possible to transform their concepts into MOF structures.

Main abstract concepts to be included in a common meta-model are relation types like single/multi inheritance, aggregation, composition, generation, attribute etc. and the foundation for the definition of domain specific descriptions (DSD). Besides their intention to be interpreted as model-based descriptions, DSD for a production environment are comparable to Domain Specific Languages (DSL) as defined in MDA [OMG 2001a; Frankel 2003] or other meta-model-driven generative approaches for software

development, for example Model-driven Software Development (MDSO, [Stahl & Völter 2005]).

A Meta-Meta-Model defines the specification infrastructure for all partial Meta-Models like a common language for the definition of model types. We call these meta-models which are defined as instantiations from the common Meta-Meta-Model Domain Specific Descriptions when they define an ontology or simple typology. Once a meta-model specifies a syntactic (e. g. graphical) notation and semantic relations for its instances, we call it diagram type to distinguish classifications from enclosed diagram meta types.

Instantiation of diagram types produces a diagram, containing e.g. a work flow definition, which in some cases can be instantiated again to form a concrete project or work flow instance.

The same concept as with use cases applies to the production control process example in figure 2 below.

3.6 Rules and Conditions

Meta-Model rules are needed to specify valid constructs within the models built under them. Meta-Models can directly specify permitted relations between specific component types. For example, a use case meta-model could specify a uses-relation type connecting actors to use cases.

Each use case model being an instance of the use case meta-model would then be allowed to create uses relations from actors (instance of actor type) to use cases (instance of use case type). Specialisations of the use case meta-model – derived via inheritance from the standard use case model – could then substructure and formalize use cases into interaction steps as this has been proposed e.g. in [Schlegel et al. 2004] to add

model semantics and formalization necessary for generative processes. This concept for creating model or diagram types becomes vital when work flow definitions have to be created and adapted by companies rather than implementing new or parameterising existing software.

Deriving important component types from a base component in the meta-meta-model separates them model-inherently and allows for a model-based interpretation by each peer system within the complete production system. Even relations, model types and item ontologies can be defined and interpreted this way to prevent users from mixing different components and systems from wrongly interpreting them.

The paradigm and methodology itself does not implicitly define or limit rule implementation to a specific concept. Rules should be defined within the meta-model and as part of the com-

mon model infrastructure used. Nevertheless, an – even partial – implementation outside the common model may be used when suitable for specific purposes.

As the strength of the approach lies in the use of the model semantics, the rule modelling approach and its engine should target and access the model structure. This is only possible when components and their relations are introduced to the rule definitions together with their object semantics.

Good candidates are object-oriented rule and condition languages such as Object Constraint Language (OCL, [OMG 2006f]), which is a side-effect free constraint language to be used in object-oriented modelling, especially UML.

Meta-Models already include definitions of available component types and possible relations between them. Using a rule language, more complex

rules including transitive dependencies and value ranges can be modelled and added to the common model. This makes it easier to restrict modelling to domain compliant models by including domain specific restrictions and characteristics into the meta-model.

3.7 Implementation-Neutral Concept

As the base concept is implementation-neutral and based on a common meta-model, it allows for easy introduction of new types of services [Husen et al. 2005; Schlegel et al. 2005], components, relations and models dynamically. This dynamic adaptation of the common model structure during run time leads to a dynamic production system completely driven by its common model infrastructure that can be dynamically adapted while the system is running productively – coping with changes of the production environment just in time of occurrence.

Only basic definitions of the meta-meta-model such as component, relation and its specialization "inheritance" are transferred to a software system. This way, the basic structure of the common model is directly reflected in the software system. All further meta-models and components are executed and interpreted by the software but do not become part of the code structure and implementation. Therefore, adaptations and extensions of the model will not lead to changes of the core software and keep the model dynamic and adaptation-friendly.

However, where code-based interpretation or treatment of model elements becomes necessary, a listener-based plug-in concept provides access to specific model elements in case of additional handling becoming necessary. Invocation of the interpreter is triggered by type and condition, e. g. event class "ma-

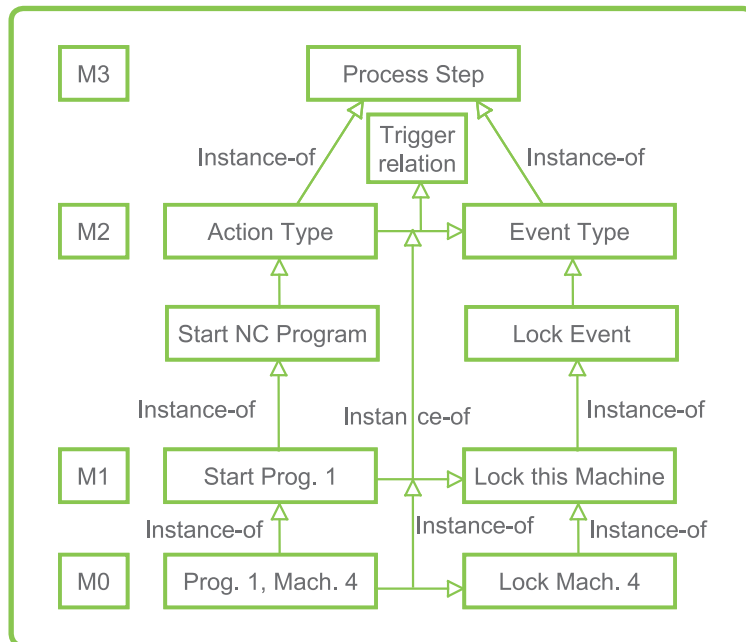


Figure 2: Sample Structure of Common Model on Meta Layers M3 to M0

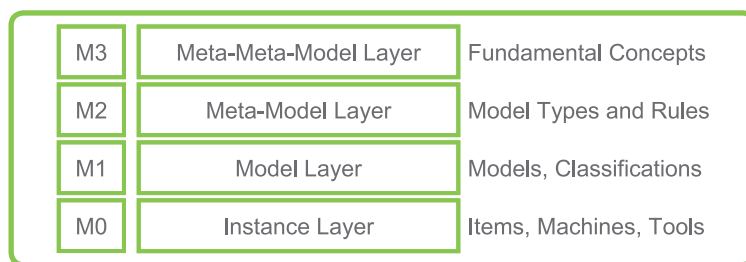


Figure 3: Four-Layered Meta-Model Structure Similar to MOF, See for Example [Frankel 2003]

chine stop event" with context "machine m1" could be used to implement special handling for machine stops of machine m1. Only an additional listener would have to be implemented – the rest of the system could remain unchanged, still sending the same events and executing the same model as before.

3.8 Relation-Meta-Components

Even core elements like relation types are defined as components in the meta-meta-model through relation-meta-components. Relations are then realized as instances of relation-meta-components, which are then recursively used to describe the inheritance structure. A clear separation of abstraction layers for relations is not possible, because of their application in all abstraction layers.

The concepts described are independent of a concrete implementation, representation or format. An agreement on such a basic meta-model allows for exchanging models easily using adaptors between completely different formats and tools. For example, an OWL-import containing additional identity information would connect all tools using these concepts to the system, which are capable of exporting OWL. The same applies for other current and future formats, which are capable of transferring object-oriented models.

4 Semantic Message Routing

Semantic message communication is one of the key features of the system currently being developed. Semantic annotation and content-based routing will enable loose coupling between components and flexible extensibility even at run time. For example, additional sensors might be attached to a machine during operation of the manufacturing plant.

By semantic annotation, full compatibility can be maintained throughout different protocol versions. Furthermore, semantically encoded performance indicators or maintenance data can be routed along the same communication infrastructure.

Finally, it is crucial to keep the communication running even in case of local disruptions or machine shut downs due to maintenance works or failures. Therefore, it is essential to establish a message based communication backbone that guarantees message delivery. Hence, the semantic message routing needs to be running as a distributed system in order to avoid single points of failure.

However, even a semantics enabled communication infrastructure needs a predefined protocol that allows later extensions by self-describing data fields. As underlying transport protocol, SOAP is an obvious choice despite of its large overhead of meta data, since it supports self describing mechanisms and provides asynchronous message conversation. Above SOAP, an extensible, dedicated application protocol is being developed.

The new XML-based protocol that will be capable of transferring messages between any of the aforementioned components will consist of a message header and a message body.

The header contains all information necessary for the message transport, i.e. routing information, request types and sender addresses. Furthermore, as in any distributed network environment, a synchronization of local clocks will be effectuated over the message header. Finally, a publish-subscribe mechanism and other assistance functions will be localized in the header part of the messages. Hence, the service-oriented paradigm of separating interfaces from implementations is kept alive in

our system, allowing for versioning and maximum flexibility.

The message body contains a recursive data structure representing a linearisation of the object-oriented model presented before. Each recursion is refining the previous one. Hence, everything is an object, i.e. a machine. A tool can be attached to the machine, and a sensor can be attached to the tool and so on. Each entity can be parameterised by a series of attributes, such as its placement, its maximum frequency or its measuring range. Note that not only static parameters characterizing the machines are transferred, but also current run time data such as the present revolution frequency.

Furthermore, the data structure contains events and error codes that are useful for corrective action and the provision of intelligent system behaviour. By the recursive structure, the XML based application protocol is extensible at run time, which was one of the requirements since local shut downs of machines for maintenance are common in any production environment.

Structural Definitions like [Vyatkin et al. 2005] can serve as a basis for classifying hardware and software used in production environment.

5 Decentralised Concept of the INT-MANUS Control Platform

The approach described is used for developing a Smart Connected Control Platform (SCCP, [Schlegel et al. 2007]) in the European Research Project INT-MANUS [Schlegel & Müller 2005], based on the object-oriented paradigm and meta-model. It connects machines, terminals, portable devices and other processor-based components through a semantic messaging framework.

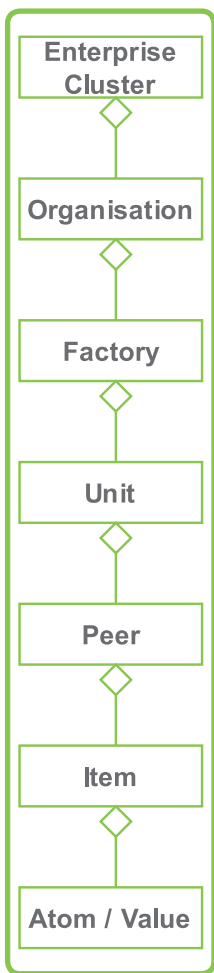


Figure 4:
Sample Aggregation
Hierarchy of Entities
of Production

5.1 Semantic Production Service Bus

A messaging infrastructure provides the foundation for message-based data transmission, which is augmented by a peer-to-peer infrastructure driven by agents for connecting all system components in the factory on a Web-Service base and exchanging model-based information.

In analogy to the Enterprise Service Bus (ESB) in general Service Oriented Architecture (SOA, [Erl 2005; Jammes & Smit 2005]), we call this bus infrastructure Semantic Production Service Bus (SPSB). While on the basic level SPSB has many similarities with the Enterprise Service Bus but requires not only advertisement of services and their connection on run time but also message-based communication relying on a strong ontological classification model.

Besides the web-service infrastructure, each message contains partial models referring to the common model infrastructure. For example a "job done" event from machine m2 is classified as a notification kind of event. With machine m2 being an instance of milling machine type connected to cell instance c1, the system can conclude that the job of this machine (instance) is accomplished and a milling machine (type) is available for the next specific task. This will notify all peers waiting for a milling machine to be available.

Interpreting this model information messages can be routed according to their type and classified destinations using the semantic message routing system as described above.

Classified sources and destinations can be identified by their attributes but also by contexts comparable to name-spaces: machines integrate their sensors and parts, cells integrate machines and terminals to form a cluster or production line within

the factory, which in turn aggregates cells and production lines to a bigger concept. In a further step it would be even possible to cluster factories or buildings to companies and these again to virtual companies connected to the market with its suppliers and customers.

The message-based system allows easy redundancy without reprogramming the system. A duplicate agent can subscribe to the same message types to share the same model or instantiate an update-listener at secondary agent copying changes of the primary one.

Cascading data is another inherent approach for consistent redundancy. Machine agents interpret and aggregate data of their machines. Cell agents operate listeners for all machines belonging to the cell, while production lines and whole factories operate listeners to receive data from underlying cells.

In case of network split, messages can be stored until they are delivered or have expired. This way any network disconnection can be handled as long as required model information is available redundant within the disconnected cell.

Logical groups like cells can of course be defined intersecting each other, for example if one machine is used by more than one production line or capabilities of a machining centre relate it to multiple functional groups.

B. Semantic Technology in the INT-MANUS platform

Different system components of the INT-MANUS platform are integrated using the distributed platform and common model.

A Distributed Model Repository is integrated into the platform for synchronizing the common model and all its components once parallel changes during a network split or communication problem e.g. due to latency has occurred. Changes in the model

can also trigger typed messages and events.

The Scheduling and Planning Module uses type information to relate classes of parts and process steps to machines and capacities allowing for seamless transition to new variants produced with the existing or (partially) additional equipment.

Learning Modules use the model information to draw conclusions from type-dependent actions also for similar components, situations etc.

Machine data is sent to the flat Data Repository for storage, aggregation and interpretation. All values are model encoded before they are sent to the repository over the platform. This means that all data is also directly related to the common model and during transportation can be routed to all type-based listeners in the same manner as other semantic messages.

This data together with model information can be used to run Data Interpretation using statistical and semantic algorithms for drawing conclusions, for example extracting possible causes for machine failure and detecting patterns in system behaviour.

Although the system is decentralized, a Security Module grants access by role-based authorization together with identity management through public-key encryption technologies.

Also platform user interfaces can profit greatly for the ontological definitions of users, roles, human-machine communication channels and environment. [Beinhauer & Schlegel 2005a; Beinhauer & Schlegel 2005a]

Connection of external systems like ERP can be achieved through transformation by a message translation gateway which transforms information contained in messages into the specific formats and actions of the ERP layer or other systems outside the platform.

6 Semantic Work flows

In the previous paragraph, the implementation of the decentralized control platform has been presented. In particular, the components for semantic data analysis have been introduced. In the sequel, it is exposed how the dynamic run time behaviour is implemented in INT-MANUS. The foremost task is the composition and coordination of work flows across the distributed components attached to both the enterprise service bus and the production service bus. In order to get an execution framework for business process automation and automated production work flows, a run time environment has to be provided that drives the business services attached to the ESB and triggers the production steps according to the component-spanning process descriptions, similar to semantic web services concepts in automation [Lastra & Delamer 2006].

In INT-MANUS, one process repository each for the business process level as well as for the working procedures have been introduced, that contain all information and run time behaviour of the production steps to be performed. The processes are represented and stored in an abstract manner, since the binding to concrete service implementations is subject to the respective service buses. Hence, the INT-MANUS architecture allows the externalization of process descriptions and the modification thereof without touching the services wrapping sensors and actuators. Such chains of service invocations are performed by a run time engine that processes the commands according to the description held in the process repository.

However, as demanded in the beginning, a link between both levels is enabled by propagating events from the production level to the business level and vice versa. Events being messages as

well, the connection between both levels is subject to business rules as well.

In some cases, a centralized architecture may be desirable, for instance in case of an emergency stop of a machine. For this purpose, a dedicated message type has been introduced that supersedes any other messages held by the SPSB.

Altogether Web 2.0 and the semantic work flow concepts pave the ground for a Production 2.0 paradigm in future production – from static, functional models and data structures to flexible, semantics-based work flows in production systems.

Creating variants of work flows as easily possible by applying the object-oriented inheritance to processes.

Even changes of a whole infrastructure of variant work flows are possible by automatically applying changes of a general process to its specialisation.

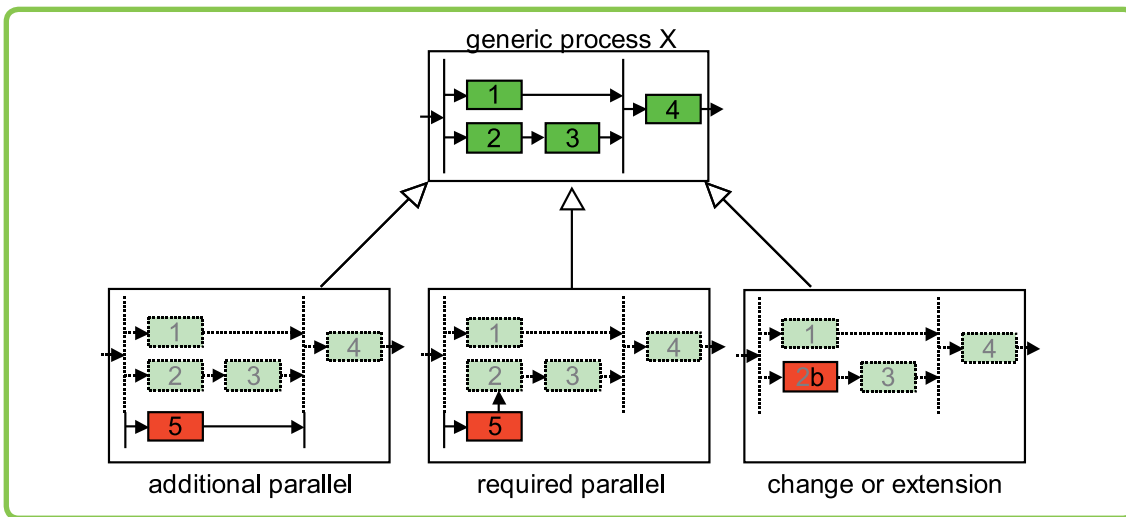


Figure 5: Process Variants by Inheritance

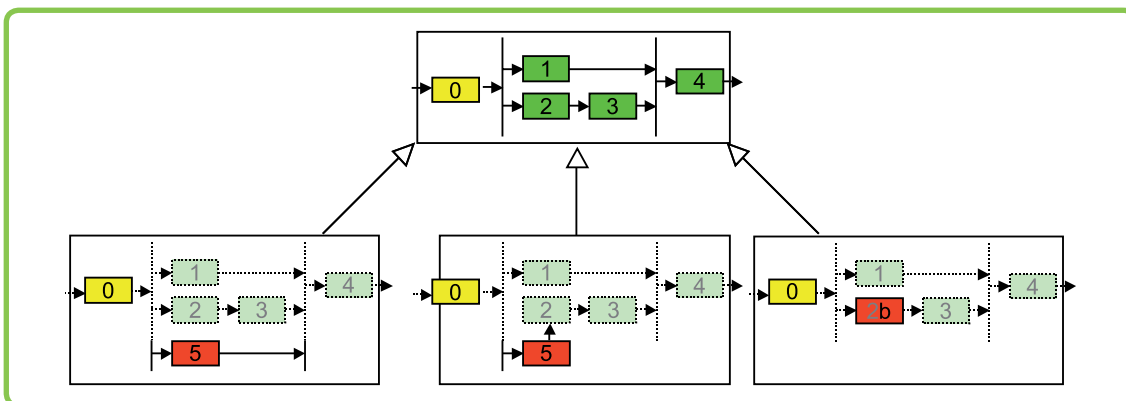


Figure 6: Cascading Changes in the Inheritance Hierarchy

7 Conclusions and Outlook

Production companies experience a high demand for rapid adaptation to new situations, conditions and requirements. Targeting rapid reconfiguration, the approach and realisation presented shows how object-oriented concepts can advance production systems towards higher flexibility. The centralized integrative approach of Manufacturing Execution Systems (MES) already demonstrates the advantages of integration, connecting formerly separated data sources through common data structures, user interfaces and – partially – work flows, connecting the "upper" layer of Enterprise Resource Planning (ERP) and Production Planning Systems (PPS) to the "lower" shop floor's automation capabilities and real time values. Using object-orientation and a clearly decentralized approach provides the system with model-based reconfiguration and independency from a central system as well as scalability. Inherent redundancy and a powerful, message-based communication infrastructure allow for dynamic integration of new peers and partial systems to the Production Service Bus (PSB).

An integrative ontology and system concept allows for handling the complete production in an object-oriented manner dynamically through model and service orientation. Agile, decentralized concepts based on a message-oriented, semantic communication and work flow infrastructure lay the ground for a new class of production systems avoiding centralized bottlenecks by applying an Internet-like concept. The object-oriented production described does not only target design and development of components in a production system. It integrates and enhances existing paradigms and technologies from other domains to converge to a new generation approach for production systems to make them flexible, dynamic and fail safe.

8 References

- Beinhauer, Wolfgang; Schlegel, Thomas (a): *User Interface for Service Oriented Architectures*. In: *Intelligent Production Machines and Systems*, pp. 129–134, 2005
- Beinhauer, Wolfgang; Schlegel, Thomas (b): *User interfaces for service oriented architectures*, CD-ROM. *Proceedings of HCI International 2005*, Erlbaum: 2005
- Bergin, J.: *What IS Object-Oriented Programming – Really?* <http://csis.pace.edu/~bergin/papers/loop.html>, viewed 28 June 2006
- Erl, T.: *Service-Oriented Architecture. Concepts, Technology, and Design*. Prentice Hall, 2005
- Frankel, D.S. (a): *Model Driven Architecture – Applying MDA to Enterprise Computing*, OMG Press, 2003
- Frankel, D.S. (b): *Software Industrialization and the New IT: A Perspective on MDA*. D.S. Frankel, J. Parodi (Eds.): *The MDA Journal – Straight from the Masters*, Meghan-Kiffer Press, 2004
- Gomez-Perez, A.; Fernandez-Lopez, M.; Corcho-Garcia, O.: *Ontological Engineering*, Springer, 2004
- Grose, T.J.; Donex, G.C.; Brodsky, S.A.: *Mastering XML. Java Programming with the XML Toolkit, XML and UML*. OMG Press, 2002
- I*PROMS POM Cluster: *Deliverable D7.5: – Research roadmap covering all research areas* http://www.iproms.org/filestore2/download/633/iproms_D7.5_POM_roadmap_edited_2_06.pdf, 2005
- Jammes, F.; Smit, H.: *Service-Oriented Paradigms in Industrial Automation*. *IEEE Transactions on Industrial Informatics*, Vol. 1, No. 1, pp. 62–70, February 2005
- Keen, Martin; Adinolfi, Oscar; Hemmings, Sarah; Humphreys, Andrew; Kanthi, Hanumanth; Nottingham, Alasdair: *Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6* <http://www.redbooks.ibm.com/redbooks/pdfs/sg246494.pdf>, IBM Redbooks, 2005
- Kletti, J. (Editor): *Manufacturing Execution System*, Berlin, Heidelberg, New York: Springer 2006
- Martinez Lastra, J.L.; Delamer, I.M.: *Semantic Web Services in Factory Automation: Fundamental Insights and Research Roadmap*. *IEEE Transactions on Industrial Informatics*, Vol. 2, No. 1, 1–11, February 2006
- Müller, S.: *Modellbasierte IT-Unterstützung von wissensintensiven Prozessen – dargestellt am Beispiel medizinischer Forschungsprozesse*. Dissertation, Technische Fakultät der Universität Erlangen-Nürnberg, Erlangen, 2007
- OMG (a): *Model Driven Architecture (MDA)* <http://www.omg.org/docs/ormsc/01-07-01.pdf>, 2001
- OMG (b): *Common Object Request Broker Architecture: Core Specification Version 3.0.3* <http://www.omg.org/docs/formal/04-03-12.pdf>, 2004
- OMG (c): *Unified Modelling Language: Infrastructure Version 2.0* <http://www.omg.org/docs/formal/05-07-05.pdf>, 2005
- OMG (d): *Unified Modelling Language: Superstructure Version 2.0* <http://www.omg.org/docs/formal/05-07-04.pdf>, 2005
- OMG (e): *Meta Object Facility (MOF) Core Specification Version 2.0* <http://www.omg.org/docs/formal/06-01-01.pdf>, 2006
- OMG (f): *Object Constraint Language Version 2.0*

<http://www.omg.org/docs/formal/06-05-01.pdf>, 2006

Patel-Schneider, P. F.; Hayes, P.; Horrocks, I.: OWL Web Ontology Language – Semantics and Abstract Syntax
<http://www.w3.org/TR/2004/REC-owl-semantics-20040210/>, 2004

Schlegel, T.; Müller, K.: Integrating human personnel, robots, and machines in manufacturing plants using ubiquitous augmented reality and smart agents. In: *Intelligent Production Machines and Systems*, pp. 189–194, 2005

Schlegel, T.; Burst, A.; Ertl, T.: A flow centric interaction model for requirements specification and user interface generation, in: *Proceedings of the 7th international conference on WWCS, bridging diversity at work*, CD-ROM, Damai Sciences, Kuala Lumpur: 2004

Schlegel, Thomas: *Integrating interaction and service work flow models by object-orientation*. CD-ROM. *Proceedings of HCI International 2005*, Erlbaum: 2005

Schlegel, T.; Karapidis, A.; Rath, H.H.; Ritterskamp, C.: *Empowering service organizations by integrating creativity and knowledge management into the service development process*. *Proceedings of HCI International 2005*, Erlbaum: 2005

Sirin, E.; Parsia, B.; Grau, B. C.; Kalyanpur, A.; Katz, Y.: *Pellet: A practical owl-dl reasoner*. *Journal of Web Semantics*, Elsevier, pp. 51–53, 2007

Stahl, T.; Völter, M.: *Modellgetriebene Softwareentwicklung – Techniken, Engineering, Management*, dPunkt, 2005

Schlegel, T.; Srinivasan, A.; Foursa, M.; Bogen, M.; Narayanan, R.; d'Angelo, D.; Haidegger, G.; Mezgar, I.; Canou, J.; Sallé, D.; Meo, F.: *INT-MANUS: Interactive Production Control in a Distributed Environment*.

In: *Proceedings of HCI 2007*, Volume 4, LNCS 4553, 2007

van Husen, C.; Meiren, T.; Schlegel, T.: *Service engineering for IT enabled service products*. CD-ROM. *Proceedings of HCI International 2005*, Erlbaum: 2005

Vyatkin, V.V.; Christensen, J.H.; Martinez Lastra, J.L.: *OOONEL-DA: An Open, Object-Oriented Knowledge Economy for Intelligent Industrial Automation*. *IEEE Transactions on Industrial Informatics*, Vol. 1, No. 1, pp. 4–17, February 2005

Wolf, T.; Decker, S.; Abecker, A.: *Unterstützung des Wissensmanagements durch Informations- und Kommunikationstechnologie*. In: Scheer, A.-W.; Nüttgens, M. (Hrsg.): *Electronic Business Engineering / 4. Internationale Tagung Wirtschaftsinformatik 1999*, Heidelberg: Physica-Verlag, 1999, pp. 746–766



Dipl.-Inf. Thomas Schlegel
Thomas.Schlegel@iao.fraunhofer.de

Thomas Schlegel is Editor in Chief of VIMation Journal and senior researcher at the Fraunhofer Institute for Industrial Engineering (IAO) since 2002. He received his degree in Software Engineering from the University of Stuttgart (Germany) and worked in Companies like HP, Agilent Technologies and Daimler. Thomas Schlegel is head of the European virtual institute VIMation, Cluster Leader for Production Organisation and Management and member of the Executive Board in the European Network of Excellence I*PROMS.



Dipl.-Phys. Dipl.-Inform. Wolfgang Beinbauer
Wolfgang.Beinhauer@iao.fraunhofer.de

Wolfgang Beinbauer studied nuclear physics and computer science in Darmstadt, Germany, Grenoble, France and Singapore. After graduating in physics, he worked at the European Synchrotron Radiation Facility as accelerator physicist in 1998. After his return to Germany and his graduation in computer science, he joined Fraunhofer Institute for Industrial Engineering in Stuttgart, where he is currently head of the business unit Web Application Engineering.



Dr. Ing. Fabrizio Meo
f.meo@fidia.it

Dr. Ing. Fabrizio Meo, born 1959 in Naples, graduated in 1983 at the University Federico II in Naples, in Electronic Engineering. Employee as software developer in Comau S.p.A. (1984–1986). Employee as software developer in FIDIA S.p.A. since 1986. From 1989 to 2001 manager for development of software for FIDIA Numerical Controls Division. Since July 2001 Research Technical Director for FIDIA. Since July 2003 Director of Research for FIDIA. Since January 2007 Director of Numerical Control Unit for FIDIA.