UML-PA AS AN ENGINEERING MODEL FOR DISTRIBUTED PROCESS AUTOMATION

Uwe Katzke
Birgit Vogel-Heuser

Automation and Process Control Engineering
Faculty of Electrical, Information, and Media Engineering, University of Wuppertal
42119 Wuppertal, Germany

Abstract: Common engineering approaches and modeling approaches from software engineering are brought together. For the domain of process automation an implementation oriented approach for an object oriented software development for heterogeneous distributed systems is introduced. Model elements for control are added to UML as well as small-scale patterns for plant automation. Besides large-scale patterns are introduced as well as implementational models. The adoption of UML regarding applied diagrams and stereotypes for process automation will be introduced and evaluated.

Keywords: control applications of computer, control system design, modelling, real time computer systems, programmable logic controllers, programming approaches, object modelling techniques, object oriented programming

1. INTRODUCTION

Today the development of software in plant industry is confronted with an increasing complexity of problems. Traditional engineering procedures, following a centralized cpu concept where software is developed and coded without modeling are overstrained by the challenges of growing requirements and the chances of modern, highly potent hardware.

Within the framework of the research project DisPA (Distributed Process Automation) we created an UML based media for describing, modeling and implementing distributed systems. With UML-PA (UML for process automation) a modeling language is created, which is not new in every integral part, but is a mature customization to fit the needs of automation industry. Although the upcoming standard UML 2.0 has forged progresses in modeling real time systems, an UML profile which is adopted for process automation can improve existing weaknesses in this domain. The UML-PA provides improvements to give a less ambiguous modeling language for several topics. Some of them are already published elsewhere, but there is still no integrated UML model.

2. REQUIREMENTS

An overview of requirements of process automation is listed in table 1 (Vogel-Heuser et al., 2004). The criteria can be structured regarding process requirements, automation system architecture, and project. In process automation, different kinds of processes are possible. A process automation system represents a type of process, e.g. batch, continuous, or discrete. Sometimes processes are composed of different process types. They are called hybrid, since they consist of different process kinds. These process types require different control strategies and as a result they require different modeling notation features, e.g. block diagrams or state charts.

Moreover overall requirements are real time, the integration of I/O peripherals directly via a backplane bus system or a field bus and interrupts from

the process. Furthermore it is required to describe the automation architecture and the mapping from software to hardware. Multiprocessor systems may be included as well.

Regarding an automation project, there are typically different engineers or technicians involved with different qualification levels and subjects. By that fact, the notation has to be easy applicable to a certain level for process, mechanical, and electrical engineers as well as for technicians. A more visionary requirement is to support the entire life cycle with one consistent model, but there should be an appropriate notation for each phase of the project.

Table 1 Overview of requirements in process automation

Categories / Criteria		Functionality / Notation Aspects
process	batch (continuous)	transfer functions, block diagrams, differential equation
	discrete	status model, flow chart, continuous function chart, Petri Net
	hybrid	continuous and discrete process
automation system	heterogeneous or homogeneous	distribution, communication, network / central unit
		different hardware platforms
		different software platforms
		HMI, diagnostics, no screen
	time	hard and soft real time; time and event controlled systems
	implementation	IEC 61131-3 for PLC (embedded system), Proprietary for DCS; C, C++, Personal Java, Embedded Java, RT Java, Ada95
	level of automation	In product automation 100%
Project	Qualification	Easy to handle for engineers and technicians
	System life cycle	Top down design
		Modularity, component base, object orientation
		Reusability
	Tool support	Along entire life cycle

Today plant manufacturing industry mostly installs standardized automation devices for automation systems, e.g. PLCs (Programmable Logic Controller), which are programmed in IEC 61131-3. Therefore, the transfer of modeling results into IEC 61131-3 is necessary.

The IEC 61131-3 contains languages, which follow a function-oriented or procedural-imperative paradigm. The increasing use of these languages has caused a growing dependency on the accepted standard, because existing implemented systems must be expanded and the developers are familiar with the practice of “accustomed” programming techniques. The cumulative expertise about home made modules allows a limited reuse and orientation. Without the knowledge of the experience from building modules, the acceptance of reusable modules is very low (Stützle, 2002).

For special tasks, such as safety related tasks or hard real time requirements additional automation devices may be used, such as process control computers with a real time operating system (RTOS). Furthermore different technologies for field buses (i.e. Profibus, Interbus, CANbus) and wide area networks endorse the heterogeneous structure of automation systems.

For hard real time systems, specific requirements are need to be realized. A list of implementation oriented real time requirements is depicted in table 2.

Table 2 Real time requirements (functional / implementational model)

Useful Language Constructs for Real Time Programming	Useful Description of Hardware Architecture
task dispatch transition control between different states of a task/ state diagram	connection between peripheral device and technical process
Scheduling / EDF	Process peripheral / modelling of input/ output
synchronization of tasks / Semaphores, rendezvous	Architectural description of different process computer units
task activation / event handling (timer/ interrupts)	
communication between tasks (sent/receive events)	connection between different computers/ network

The aspects of real time development like reactivity, multi-threading, time-based behavior and real time environment are needed to be met. In the past, efficient and machine-intimate but Gordian programming was necessary to fulfill these requirements. Actual controllers show better performance then their predecessors.

Prior investigation compared modeling techniques for distributed process control engineering (table 1). The research analyzed UML and Idiomatic Control Language (ICL) regarding cognitive models and user acceptance (Friedrich et al., 2003).

3. NOTATIONS FOR DISTRIBUTED SYSTEMS IN PROCESS AUTOMATION

Many existing notations provide solutions for isolated requirements. Indeed no established notation

fits all of them. UML has advantages concerning its applicability across different development phases or the degree of familiarity among developers as well as users, which are also of value in embedded software engineering. But the main benefit is based on its extensibility constructs. It is possible to build specialized profiles for specific domains.

UML 2.0 specifies a performance and a time specification, useful for constructing real-time systems. But it has no formal semantic as Berkenkötter et al. (2003) analyzed. Licht (2004) proposed the integration of timed automata, which seems to be a very promising approach, regarding comprehensibility for engineers and formal specification.

Giotto is a programming language for embedded hard-real time control systems with periodic behavior (Henzinger et al., 2003). The concepts for scheduling and timing aspects need to be evaluated. Giotto worked also on distributed systems, but didn't focus on process automation specific requirements or large scale software for plants.

Ptomely (2004) facilitates functional description by providing various models of computations and hence it allows different domains, but up to now process automation is not included.

The structured analysis offers notations and modes of operation for a top down design of automation systems. This concept follows a strategy of successive refinements. It identifies objects, but it has neither a mechanism nor a notation for generalization...

For automatic code generation, a prototype has been developed, which generates IEC 61131-3 code (ST and SFC) automatically from the UML model modeled in an UML tool. This code generation prototype allows evaluating the weaknesses of standard UML by expert rating (Vogel-Heuser et al., 2004)

Within the framework of DisPA we have created an UML based media for describing, modeling and implementing distributed systems. Based on the requirements analysis, a specialized profile of the UML for process automation (UML-PA) has been accomplished and evaluated for hybrid processes and heterogeneous control systems.

4. ADAPTING UML FOR PROCESS AUTOMATION (UML-PA)

The adaption of UML for distributed control systems, i.e. modeling of controllers handles time constraints, dynamic redundancy, time triggered synchronization of distributed systems, communication structures between encapsulated modules (from the UML-RT profile) and an enlargement of the deployment diagram to describe the mapping of software and hardware architecture. All elements, which extend the UML to the UML-PA, are based on the extensibility constructs of the UML. In detail the UML-PA contains the following enhancements:

- Time constraints on architectural level
- Timed State Machines
- Communication by ports and protocols and
- Mapping between software and hardware.

4.1. Time constraints on architectural level

The UML provides several diagrams for identifying objects and their collaboration. They provide either architectural design (i.e. deployment diagrams) or simple scenarios with real time requirements, i.e. sequence diagrams of UML 2.0 (OMG, 2004), MSCs of the ROOM notation (Fuchs, 1998). During design many time dependant decisions have to be taken. Engineers need to determine the power of necessary devices as well as their number. Each device will fulfill requirements of other connected devices and their contained processes. Concurrently each device has a limited capacity for fulfilling the claimed requirements.

The load of a certain component (i.e. a field bus, fig. 1) is partly given from each connected neighbor. Additionally real time requirements raise the charge. Distributed systems allow parallel execution on different devices.

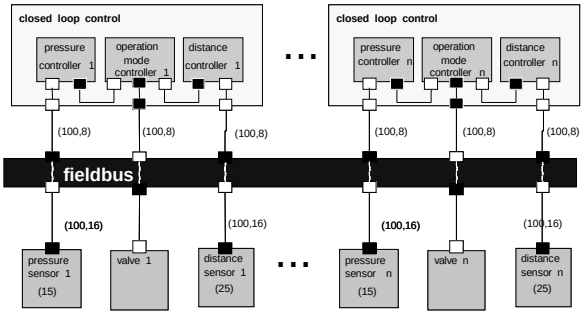

Fig. 1 Communication requirements stressing the field bus

Most single devices operate only sequentially. The different requirements have to be organized by scheduling. The Timing Diagram of the UML 2.0 delivers a scheduled view as a scenario. It is designed from estimated behavior by the experience of engineers. It gives an advice for the composition of the necessary devices. A global and traceable view need to be invariable. This is given by timed state machines.

4.2. Timed State Machines

The UML provides only fuzzy defined constructs for the description of state machines. Especially the predefined expressions for real time behavior are poorly defined. Namely the events *when* and *after* are available in the standard UML to define time demands, but there is neither common interpretation of these keywords nor a notation of its parameters available. There is no language to describe actions and conditions, which can be attached to states and transitions.

Therefore the UML standard for these constructs is constrained to regular expressions in the UML-PA. Guard conditions in UML-PA are described by well

defined logical expressions. Actions are either calls of class operations or assignments of logical, arithmetic or string expressions. Thus each action and condition is describable through compositions of simple elements. Their time behavior can be calculated quoted through worst case estimators.

The procedure of whole state machine can be investigated by starting from the intra state behavior Kardos and Rammig (2004) have ascribed the semantic of inter state behavior of UML state machines to the SDL. Events for timers are implementations of signals between SDL agents.

Based on the concepts of timed automata (Licht, 2004), certain predefined events and constraints are defined to provide a clear notation for formulating a definite real time behavior.

The UML-PA combines both approaches to timed state machines which offer a clear media for describing real single and circular time events.

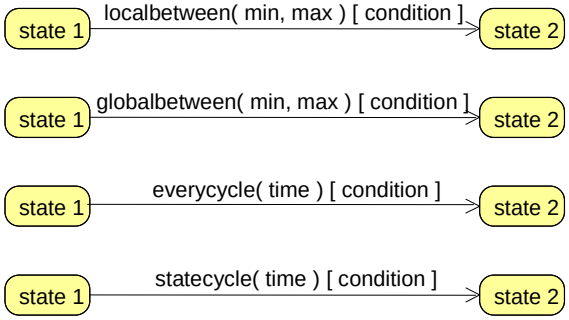

Fig. 2 Time events in UML-PA

Fig. 2 shows four different time related events of UML-PA. The semantic of each event is defined by mappings to UML state machines with an integrated timer. The events *globalbetween* and *localbetween* allow the definition of timeframes. Lower and upper bounds determine the available time for state internal behavior. The events everycycle and statecycle describe cyclic occurrences of timer events. The difference is in the constraint of both events. Whereas statecycle events expire on occurrence, if the origin state is not reached, everycycle events cause an error in this situation.

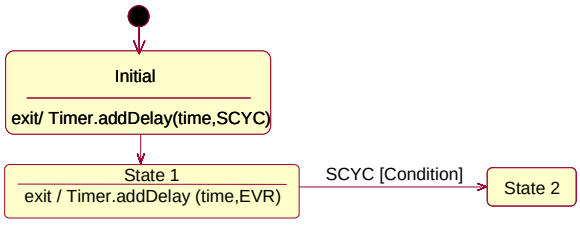

Fig. 3 Mapping of statecycle event to UML state machine with an additional timer object as an event source.

The mapping of the statecycle event in fig. 3 gives an example, how the predefined time events are ascribed to signal based events (Kardos 2004). The timer is added to the state machine without modifying the definition of states and transitions. Each time event can be substituted by a standard UML notation, but the uses of these events reduce the complexity of models by saving states and transitions.

4.3. Communication by ports and protocols

The need of well defined communication is evident in distributed (automation) systems. The first steps when designing a distributed system are embossed by the identification of entities and their interaction. In a process of refinement new entities are generated and applied with additional communication links. These need to be equipped with protocols and the roles of each participant have to be defined. Therefore the UML-PA is supplemented with the extensions ports, capsules, protocols and roles from the UML-RT profile.

Capsules are stereotyped classes which are restricted by constraints. Each capsule object communicates only with other objects via its attached ports (Fig. 4). Ports are communication interfaces of encapsulated classes. Their instances are added by declaring public attributes in a capsule. All other attributes and operations of a capsule class are declared private. A connector describes the communication relation between two ports.

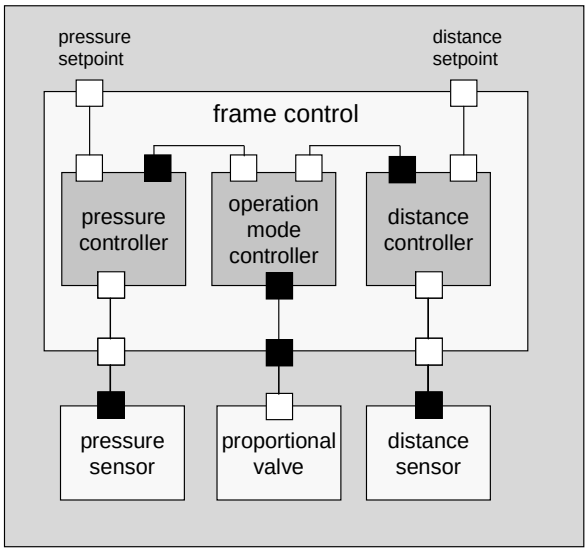

Fig. 4 First identified entities of the application

Fig. 4 shows a structure diagram for designing the communication dependencies between several entities of a single hydraulic cylinder in the application example. At this point the communication is described still incompletely, but the missing parts (i.e. protocols and the links between hardware and software) are modeled in other diagrams. The links between different controllers, sensors and actuators in UML- PA conform to the presentation of controller links in block diagrams.

4.4. Mapping between software and hardware

Deployment diagrams and component diagrams as the current media for modeling relations between hardware and software don't fit the requirements of process automation. These systems operate with widely distributed hardware.

Identical hardware is used very often at different places. It must be possible to instantiate multiple occurrences of the same hardware, to provide it with its unique record of attributes (i.e. addresses for identification) and to link it to the instance of a connector of the model.

Therefore a specialized object diagram contains model external entities (i.e. hardware entities). Predefined classes for typical hardware objects like sensors, actuators or field busses facilitate the mapping of inputs and outputs to the connectors to the corresponding hardware. The benefits of these UML extensions are demonstrated by implementing a closed loop control system which realizes the press of a fiber board plant as a prototype of an industrial application.

Classes of the stereotype <<adapter>> are defining sensors, actuators and other items, whose behavior is determined outside the application model. They are represented in the UML-PA model as capsules without operations and attributes.

5. MODELLING WITH UML-PA

The UML-PA offers a self contained model for the static construction of automation systems and their behavior. Due to the diversity of notations for different aspects within the UML, guidelines are required. The principles of the structured analysis give an assistance, which can be transferred to the notations of UML. Starting from a black box system which is arranged with its peripheral devices in a context diagram this procedure of refinement breaks a system into small parts by refining external

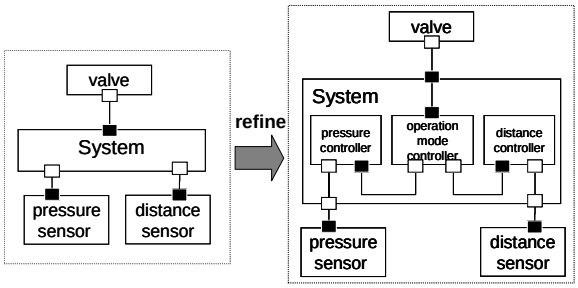

Fig. 5 Successive refinement of a capsule

communication flows. Using the UML-PA, this procedure is transferred to the refinement of capsules (fig. 5). Each capsule represents a system which can be refined in a subordinate diagram unless it is already a primitive element.

5.1. Auto reconfiguration

The abstraction level of the object oriented approach allows the construction of virtual devices. They are used to establish a service for the online reconfiguration of a system to displace defective hardware (i.e. sensors).

The reconfiguration service can be used in new or already existing models to improve the functionality (i.e. safety) of target systems. A controller pattern consists usually of a sensor, an actuator and a controller device. The failure of one of these components results in failure of the entire control chain. This safety problem can be prevented by usage of redundant devices. However, the integration of redundant devices into an existing model is one of the main challenges. Often the model structure and also already implemented code have to be changed. An inherited reconfiguration class is designed to solve this problem. It is a virtual representation of a substitute device. Any deviation is corrected by an adjustment function (fig. 6).

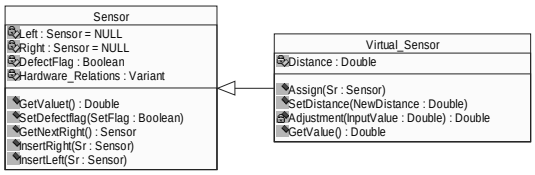

Fig. 6 Virtual_Sensor as an inherited reconfiguration class

As an example we suppose an additional sensor or an already existing is needed to improve the steadiness of a system. In this case the reconfiguration class will simply replace the primary sensor device. The reconfiguration class Virtual_Sensor instantiates the physically existing device Sensor_2, but it corrects the measurement in a way, that the system's handling of the substitute is identical to the original device. Therefore the reconfiguration class uses the mechanisms of inheritance and overriding.

6. EVALUATION

The application of a continuous hydraulic press for particle board production has been used to evaluate the UML-PA itself and the developed procedures. There each hydraulic group is controlled by one distributed control device with several closed loop controllers. Due to global safety constraints some higher-ranking global functions require a synchronized controller switch in all hydraulic groups, i.e. in each distributed system simultaneously.

The closed loop controls, implemented from the UML-PA model for press control are performed on a simulated model (Matlab Simulink) of the press and there they are confronted with a reactive behavior. Both types of systems, the process model and the controller are connected via CAN as field bus system. To evaluate reconfiguration using dynamic redundancy (to displace defective hardware) the process model could simulate faulty sensors, which require a replacement of sensors using the sensor data of a neighbor hydraulic system. Bus load is monitored as well as the compliance with timing and the functionality constraints.

7. SUMMARY

At first we find the successful development of notation elements for open and closed loop control, interlocking and redundant hardware aspects, which are part of characteristic aspects of embedded sys-

tems. We designed intuitive applicable elements on the basis of typical modeling constructs in control automation. In addition we show a possibility to bridge the gap between the function oriented paradigm used in plant automation industry (IEC 61131-3) and the object oriented approach, which is an important aspect for adoption in industry. This aspect is not at least shown by the usage of reusability.

At this point the presented parts of UML-PA may look fragmental, but the complete UML-PA will bridge the gap between process automation and computer science. The composition of these parts is the result of the detailed requirements analysis, which was made already in the forefront of the UML-PA elements' development. These requirements are based on one hand on different interviews with software developers and users (Katzke et al., 2004) and on the other hand on the evaluation of presently offered UML tools.

8. OUTLOOK

Most UML-PA extensions can be realized through annotations within such tools. Based on the UML code generator the UML-PA should be interpreted. By this it will be implemented in such a tool. To develop distributed systems the timing of bus systems needs to be included in such a model. The integration of tool simulating bus systems, e.g. Ethernet, should be coupled with an UML-PA approach.

Outside the scope of this project, but nevertheless an interesting enhancement of software, that is designed using UML-PA are the aspects dynamic reloading during runtime, automatic (re-) configuration and decoupling of software fragments. UML-PA offers promising fundaments to realize such features, due to its supported characteristics concurrency, structural independence, and immutability (by a third party). The challenge to realize the above mentioned abilities is the required communication infrastructure of the software components. Today, this is usually accomplished by communication interfaces, which are also supported by UML-PA. Reconfigurable software would need additional features like the ability to negotiate with other parties and to dynamically adapt to given needs. Therefore, continuative methods for software development are necessary.

9. REFERENCES

Berkenkötter, B., Bisanz, S., Hannemann, U. and Peleska, J. (2004), HybridUML Profile for UML 2.0, SVERT., *workshop hold in conjunction with UML 2003*, San Francisco.

Friedrich, D., Vogel-Heuser, B. and Bristol, E. (Nov. 2003). Evaluation of Modeling Notations for Basic Software Engineering in Process Control. In: *29th Annual Conference of the IEEE Industrial Electronics Society (IECON 03)*, VA, USA.

Fuchs, M., et al. BMW-ROOM, An Object-Oriented Method for ASCET, Society of Automotive Engineers, Detroit, 1998

Henzinger, T. A., Kirsch, C. M., Sanvido, M. A. A. and Pree, W. (2003). From Control Models to Real-Time Code Using Giotto, *IEEE Control System Magazine 23*, **vol. 1**, pp.50-64.

Huber, F. and Schätz, B. (Dec 2001). Integrated Development of Embedded Systems with AutoFocus. *Technical Report TUM-I0107*, TU München, Institut für Informatik, Munich

Kardos, M., Rammig, F., J. (2004). Model Based Formal Verification of Distributed Production Control Systems. *Ehrig et al. (Eds.) Integration of Software Specification Techniques for Applications in Engineering, Lecture Notes of Computer Science*, Springer, Berlin.

Katzke U., Vogel-Heuser, B. and Fischer, K. (2004). Analysis and State of the Art of Modules in Industrial Automation. *atp international*, **vol. 1,** pp. 23-31,

Licht, T. (2004). Ein Verfahren zur zeitlichen Analyse von UML-Modellen beim Entwurf von Automatisierungssystemen. *PhD Thesis, faculty of computer science and automation*, Technical University of Ilmenau.

Nickel U., Schäfer, W. and Zündorf, A. (2003). Integrative Specification of Distributed Production Control Systems for Flexible Automated Manufacturing. In: *M. Nagl, B. Westfechtel (Eds.) Modelle, Werkzeuge und Infrastrukturen zur Unterstützung von Entwicklungsprozessen*, Wiley-VCH Verlag, Weinheim.

OMG, UML 2.0 Superstructure Specification, www.omg.org/cgi-bin/doc?ptc/2004-10-02

Overview of the Ptolemy project, technical memorandum UCB/ERL M03/25, http://ptolemy.eecs.berkeley.edu, 2004-09-29

Stützle, R. (2002). Wiederverwendung ohne Mythos. *Empirisch fundierte Leitlinien für die Entwicklung wiederverwendbarer Software*, PhD Thesis, faculty of computer science, University of Munich.

Vogel-Heuser, B., Friedrich, D., Katzke, U. and Witsch, D., „Usability and benefits of UML for plant automation – some research results" accepted paper " in atp international, 3(2005), issue 1, Oldenbourg Verlag, Munich, 2005.

Vogel-Heuser, B., Fischer, K., Göhner, P., Gutbrodt, F. and Katzke, U. (2004). Conceptual Design of an Engineering Model for Product and Plant Automation. In: *Ehrig et al. (Eds.) Integration of Software Specification Techniques for Applications in Engineering, Lecture Notes of Computer Science*, vol. 3147, Springer, Berlin.