I. Troch, F. Breitenecker, eds. ISBN 978-3-901608-35-3

INTEGRATED MODELING, SIMULATION AND OPERATION OF HIGH FLEXIBLE DISCRETE EVENT CONTROLS

T. Pawletta, S. Pawletta, G. Maletzki
Hochschule Wismar – University of Technology, Business & Design, Wismar, Germany

Corresponding Author: T. Pawletta, Hochschule Wismar, RG Computational Eng. & Automation (CEA)
PF 1210, D-23952 Wismar, Germany; thorsten.pawletta@hs-wismar.de

Abstract. An approach for integrated modeling, simulation and operation of complex discrete event systems is introduced and illustrated by the example of a re-configurable sensor based robot control. It is discussed how simulation models of complex discrete event systems have to be structured in the early system design stage in order to extend them to model based control programs for the operation stage stepwise. One important task in this context is to differentiate between control and process parts clearly. Moreover, the description of task-based control programs is discussed and the well-known System Entity Structure and Model Base (SES/MB) framework is adapted for specification of high flexible controls and stepwise generation of task-based control programs at runtime. Finally, a prototyping development is presented.

1 Introduction

Modeling and simulation of discrete event dynamic systems is a well established method in engineering science. Discrete event simulation (DES) is used in the *design phase* of complex systems to compare different system structures, layouts and control strategies [19]. During a DES project the system model and control strategies are implemented in a simulation model using one kind of simulation software. Simulation in that phase can prove the required system performance including used control strategies before the system is built. With completion of the design phase the subsequent *automation phase* starts. It aims to develop and implement the process control system. Usually, the outcome of a DES project is a document with a description of structure, layout and control strategies. The conventional procedure then re-implements control strategies into a specific process control software system, whose features are often quite limited. The results are increased programming effort, a high probability of hidden programming errors and communication problems between system engineers and control engineers. In contrast to control engineering of embedded applications [22] there is a gap between design phase and automation phase up to now. Other research has also clearly identified this gap, for example in container terminal automation [21]. Anderson [2] summarises that this software development process may consume up to 40 percent of total system costs. Re-use of software from the preceding design phase is one approach to reduce the costs.

Current approaches dealing with the identified gap are driven by the design phase where detailed simulation models have been developed and implemented to analyse process control strategies. A prerequisite for efficient control strategy analysis and model re-use for process control is appropriate structuring of the simulation model. This research proposes the separation of a simulation model into two general sub-models, a *system model,* also called *process model,* and a *control model*. The system model maps the dynamic of the elements of a real or planned system and their couplings. It receives input values from the control model and sends state values to the control model. There are two different approaches [1] to realise a simulation model for process control: *(i) explicit control synthesis* by automatic code generation and *(ii) implicit control synthesis* by extension of the simulation model with a process interface. However, today there exist only few commercial code generators for specific embedded systems. There are no compilers available to translate simulation models to execution code for Programmable Logic Controllers (PLC) or controllers for industrial robots. This research is based on an implicit control synthesis.

One of the first approaches to extend a simulation model for process control has been published by Zeigler and Kim [23]. The approach is based on the DEVS-theory (Discrete Event System Specification) developed in the 1970s to formally describe a discrete event system. In [18] an application is presented using the real-time extension of DEVS to develop a controller for an embedded system. In [9] DEVS is used in the context of manufacturing systems. Other research is based on a more intuitive approach based on the characteristic of simulation systems. In [15] an approach is proposed using the simulation software *Arena* to link simulation software with a physical process. Another approach is initiated in [21] using the simulation software *Simple++* to develop and operate control strategies for a container terminal application. Common in all these projects is that the system model is replaced by a process interface for communication with the real physical system and the control model acts as a process controller.

The approach presented in this paper is based on the work in [10, 12, 13], where the system model remains part of the process control system. Hence, this approach is able to support model based control methods in analogy to

the state observer approach in continuous feedback control theory. Basics of this approach and extensions for a simple task oriented specification of the control model are described in section two. Section three introduces an extended task oriented control view based on the *supervisory control* approach and *adaptive control methods*. High flexible and re-configurable systems are characterized by a limited range of system autonomy [3, 5] and the supervisory control has to keep the system within limits [8]. In such applications it may be necessary to adapt control parameters or the control structure at runtime. We suggest the *System Entity Structure (SES)* and *Model Base (MB)* framework according to [20, 24] as meta-modeling tool for the specification and management of various control structures and parameters. In this context modified graph analysis and model generation algorithms are developed, which support stepwise graph analysis and comprehensive reconfiguration of the control model at runtime. Fundamentals of this research are demonstrated by the operation of a high flexible sensor based industrial robot control. Finally, the development of a prototyping environment is presented briefly.

2 Extending a simulation model from design stage to an event-based control system

2.1 General approach

During the design phase it has to be strictly distinguished between *system model* and *control model*. The system model contains elements (e.g. buffer, server, etc.) representing real process devices. The elements are connected as the real "material flow". Each element has internal variables to save the current state, such as number of jobs in a buffer, processing progress, etc. State information is used in the control model and for additional features, such as visualisation or data recording. The control model generates control signals, used in the system model to control activities and the material flow between elements. This general structure is presented in figure 1a.

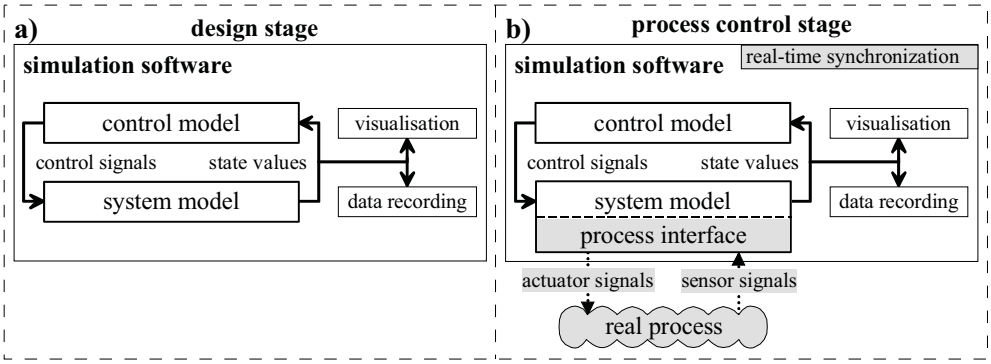

Figure 1. Extending a simulation model developed in the design phase (a) to a model-based process control system (b)

Two significant extensions on the simulation model and simulation software are necessary to act directly as a process control system. The changes are highlighted with a grey background in figure 1b. The simulation model is extended by a process interface. It is used to receive sensor signals from the real process and to send actuator signals to the real process. In this context some changes in the system model are necessary. System parameters such as delay times of machines are replaced by real process events, which are received or sent via the process interface. During the design phase a simulation model is executed as fast as possible. Thus the second extension synchronises the simulation engine with real-time. That means a real-time clock drives the simulation model. A prerequisite for a successful synchronisation is that the execution of the simulation model is faster than real-time. Of course, in real applications a process interface can be implemented modular and hierarchical in different layers, described in more detail in the next subsection. Moreover it is also possible to test process interfaces in design phase. A possible approach is the development of process visualization with a real process interface protocol that is running on a separate PC with appropriate interface cards as described in [12, 13]. The next subsections will detail the approach.

2.2 Development of system model, control model and process interface

Figure 1 shows that the simulation model consists of a system model and a control model. Both models are discrete event dynamic systems and they can be specified in a hierarchical, modular manner according to the general DEVS (Discrete Event System Specification) concept of atomic and coupled systems as described in [24]. According to this theory atomic systems contain a dynamic description, i.e. the definition of state variables, state transition functions etc. Coupled systems define a structure comprising a set of sub-systems and coupling relations. Both system types can be connected using input and output ports (variables) and they communicate via events. Even though DEVS is a powerful formalism, it is not widely accepted in the engineering community. Hence, we use the concept of statecharts [7] that is more common in engineering today and similar powerful like DEVS. In addition to the pure discrete event oriented nature of DEVS we use conditions of continuous signals for state transitions and event generation too. In the following, we discuss the component oriented development of the system model, the structure of the control model and the implementation of process interfaces in more detail.

Component oriented development of system model in design phase

As example component a general *(m,n)* transport node, pictured in figure 2, is used. Such a transport node may be a robot with *m* input places and *n* output places for jobs in a manufacturing system. Figure 3 shows a *(m,n)* transport node component with *m=2* and *n=2* of a system model. The component has one material flow input port and output port *(IN_job, OUT_job)* and one input and output port *(IN_control, OUT_state)* for communication with the control model according to figure 1. Because, the component at this stage has no process interface we call this part *Transport_node_High_Level*. Figure 4 illustrates the *Transport node High Level* system dynamics specified in a statechart.

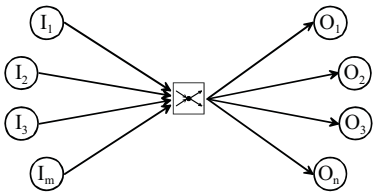

Figure 2. Structure of a general *(m,n)* transport node

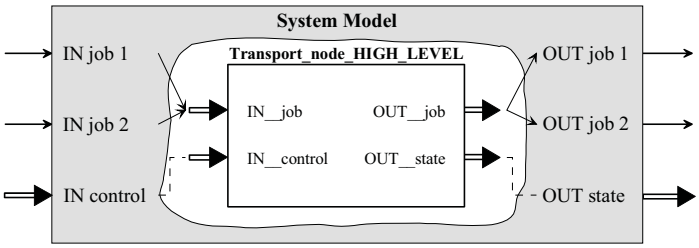

Figure 3. (m,n) transport node high level component with m=2 and n=2 in a system model in design phase

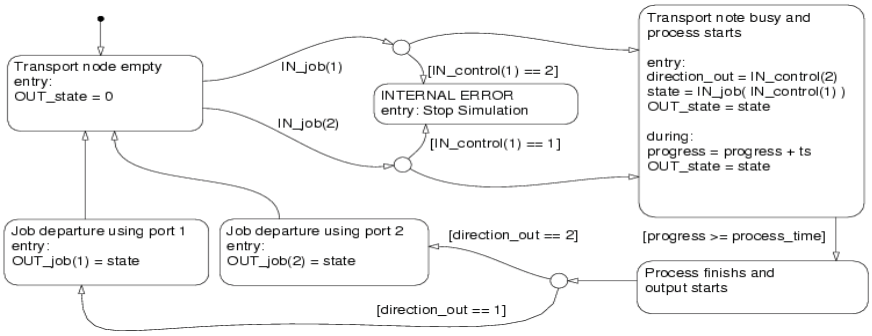

Figure 4. Statechart of (m,n) transport node high level component with m=2 and n=2

An incoming event on job input port *IN_job* changes the transport node from *"empty"* to a *"busy"* state and induces an output signal to the control model (*OUT_state = IN_job*). Concurrent with the detection of a job at the *IN_job* port a control signal is received on *IN_control* port. This signal contains the input and the requested output direction of an incoming job. Comparison of detected (*IN_job*) and received (*IN_control*) input direction is used for an internal error checking. The state change from *"empty"* to *"busy"* starts the process in the transport node. Optional transport directions require flexible values of the *process_time* depending on the current job type and routing. When the process has ended the output of the current job starts using the specified output port. After the job leaves using *OUT_Job* port the new transport node state is *"empty"* and an appropriate signal is sent to the control model (*OUT_state = 0*).

Our approach is based on modular modeling and a strict separation of system and control parts. That means the whole system model has to be modelled component oriented, whereas components may be grouped together in coupled systems. Figure 5 shows a section of simulation model in design phase comprising buffer and server. The server is modelled using a simple *(1,1)* transportation node. The *lower part of figure 5* represents a section of system model. Connections between the system model components describe the event oriented material flow and are used to transfer material information (e.g. job type, manufacturing progress, quality etc.). The blocks s*_buffer*, s*_server*, *c_buffer* and *c_server* are necessary to achieve the connections of the system model and control model. The blocks s*_buffer* and s*_server* send the state values to the control model and the blocks *c_buffer* and *c_server* receive the control signals.

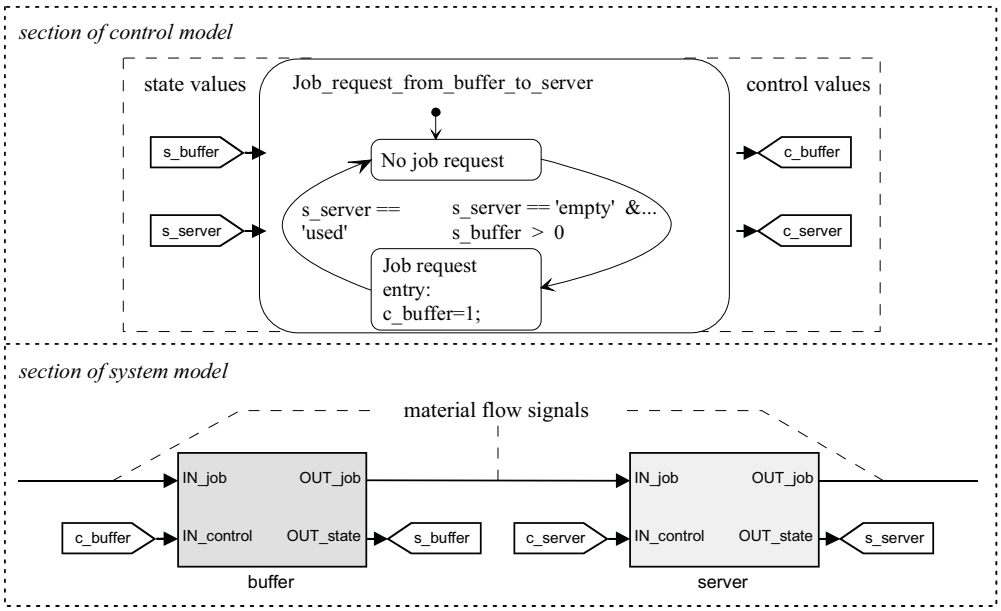

Figure 5. Section of a simulation model structure in design phase

Developing the control model

The *upper part of figure 5* represents a section of control model. The task of the pictured control component is to generate a control signal (*c_buffer=1*) to allow a job to leave the buffer. This control signal is generated if there are jobs in the buffer (*s_buffer>0*) and the server is in state *"empty"*. In this example the control model section consists only of two states and two state transitions and it defines no time dependent behaviour. Hence, the final control program runs in a real-time synchronized simulation environment it is obvious possible to define active monitoring tasks (e.g. checking if operations are fulfilled in defined time limits) as described in [18].

In real applications the control model may have a very complex structure. Then it is advisable to build it up hierarchically. Dilts and Pelts et al. [4, 14] discuss different hierarchical process control structures. Our research experiences led us to favour the *proper hierarchical form* and the *modified hierarchical form* to implement the control model. Figure 6 shows the *"proper hierarchical form*" which consists of a Leading Control and several Group Control units. In the *"proper hierarchical form"* communication is only allowed between the Leading Control and the Group Control units. The *"modified hierarchical form"* allows in addition communication among the Group Control units. Using the "*proper hierarchical form*" Group Control units receive control signals from the Leading Control. These control signals define which strategies should be used in parallel branches (e.g. priority based part routing, machine priority or batch size constraints). Concurrently Leading Control receives state information from the Group Control unit. Allocation of capacities and control of jobs inside a Group Control unit is assigned by its own logic. Each Group Control unit and also the Leading Control are directly responsible for a specified number of components in the system model.

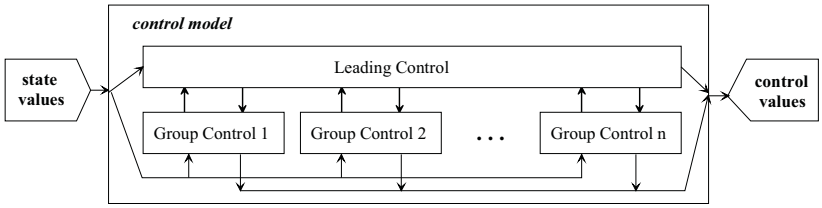

Figure 6. *"Proper hierarchical form"* of a control model

Development of process interfaces and the process control phase

The existing simulation model from the *Design Phase* should be used for process control by adding a process interface. The process interface provides in general the ability to evaluate sensor signals and to generate actuator control signals. Our approach is based on a distributed structure of the process interface. In general, each atomic or grouped system model component has its own adapted process interface to link the corresponding devices of the physical process. Of course, the process interface can be structured hierarchically and lower levels may be shared by different components. The upper level of a process interface to a real process device is attached to its corresponding system model component. The introduced transport node, as an example of a system model component, was termed *High Level* in design phase, the attached process interface is termed the *Transport node Low Level*. The key principle in this research is illustrated in figure 7. It describes the coupling of the system model functionality with a process interface using the implemented *(m,n)* transport node example with *m=2* and *n=2*.

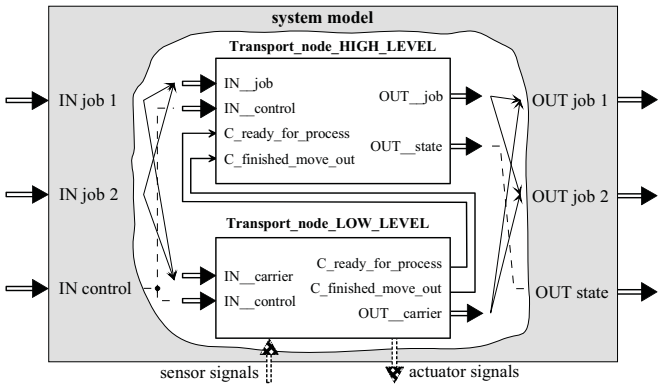

Figure 7. Coupling of *High* and *Low Level* part of a system model component (example: *(m,n)* transport node)

The delay times in the transport node used during the *Design Phase* in *High Level* must be substituted by real process events. Therefore ports are integrated in *High Level* to receive real process events from *Low Level*. Two communication ports work as follow between the *High* and *Low Level* in the transport node element: The real process in the transport node device will be started from *Low Level*. If a sensor signals a transport carrier has left the real process device, then the *C_ready_for_process* signal indicates this event in *High Level*. This event uses the *C_finished_move_out* communication signal to update the *High Level* state.

Low Level modules largely depend on the real process devices. However, the suggested control approach with an integrated system model opens sophisticated ways for discrete event process control similar to the well known state observer approach in continuous feedback control. In complex and flexible manufacturing systems it can be necessary to know job or carrier identifiers. There are two approaches to establish identifiers:

sensors (e.g. bar code),

save and transference carrier identifiers through material and information flow.

Economic, flexibility and reliability problems make the second a better option. In figure 7 *Low Level* receives the carrier identifier through input port *IN_carrier* concurrent with an *IN_job* signal in *High Level*. When an *IN_carrier* signal arrives, the carrier identifier will be saved in the *Low Level* state. Actuator control signals are generated for the specified transport carriers and the real process devices in the *Low Level* part. When an output sensor signals a transport carrier has left the real process device, the following actions are triggered:

Low Level generates *C_finished_move_out* signal,

Low Level outputs carrier identifier,

High Level outputs job type,

The states of *High* and *Low Level* are updated.

The system control strategies have already been implemented during *Design Phase* in the *control model*. These implementations are used unmodified in the *Automation Phase* during process interface adding and subsequent in the *Process Control Phase* of a real system.

In addition to the process interface the key to control a real process using the *implicit control synthesis* approach is the *synchronisation of the simulator with real-time*. That means a specified time step in the simulator must be exactly the same time step in the real process. This is the foundation for processing times and delays. Furthermore a cyclic scan of the sensors is necessary to register all process events in time. For this work a *fixed sample time simulation clock* is used. The time synchronisation principle is based on sliding of the sample time intervals with real time intervals. This is integrated in a module that must be added to the simulator. Inherent in this principle is that the execution of one time step of the simulator needs to be faster than one sample time interval.

As mentioned in the beginning, a process interface can be implemented hierarchically. Complex physical devices have specific controllers, which we call execution controller. The execution controller of an industrial robot implements for example powerful algorithms for path planning and workspace monitoring. In Section 4 a practical prototyping environment with an industrial robot is presented and it is shown how a specific robot controller can be integrated as an execution controller.

2.3 Simple task-based controls

In the task-based approach the *control program consists of a set of different tasks*, where some tasks may be executed in a defined order or in consideration of different restrictions whereas other can be executed in any order. According to Kruth et al. [11] a task is not a simple command with a primitive action as result. A task defines a more complex goal such as the identification of an object where the result may be the object type, the

object position etc. A task-based control program necessitates a task transformer that translates a task description in executable actions of physical devices. In reverse, sensor signals have to be transformed in a task-based data format.

In the introduced simulation model based control system *the task based control program has to be specified as control model*. The task transformer has to be implemented in the process interface, i.e. in the *Low Level* part of system components. Often it is useful to integrate execution controllers for specific physical devices. Then, the *Low Levels* are responsible for the generation of traditional control commands executed by the device controllers. In the following, the specification of a task-based control model will be considered. According to the modular modeling concept a task-based control model can be composed by different basic tasks (atomic models), which may be grouped together in coupled models to subtasks and to one complete control task finally. Basic tasks and subtasks can be executed in sequence or in parallel. Sequential tasks are characterized by unidirectional couplings, which may be used for event-based activation or data exchange. Moreover, each basic task or subtask pre-defined in a model base can be parametrized when used in a control model.

Figure 8 shows an example of a simple task-based control model specified using pre-defined task models from a library. The control model in figure 8 is a detail of the robot application pictured in figure 12. The robot cell consists of an input buffer with different units and three output buffers. The robot has to sort the units according to given characteristics (e.g. the colour) in the three output buffer. Therefore, a picture-taking sensor is used to identify the units in the input buffer. The control model detail consists of parametrizable subtasks, which are executed sequentially. The subtask *Init* is responsible for the initialization of robot cell application. The task sequence *Make Picture*, *Pick Object* and *Place Object* describes the identification of one unit in the input buffer and its transport to the appropriate output buffer. The type of unit and the address of appropriate output buffer are specified by the parametrization of each subtask. Finally, the subtask *Stop* represents the end of a complete placement task. A possible parallel task is to monitor collisions with other physical devices in the cell simultaneously to the characterised placement task [13].

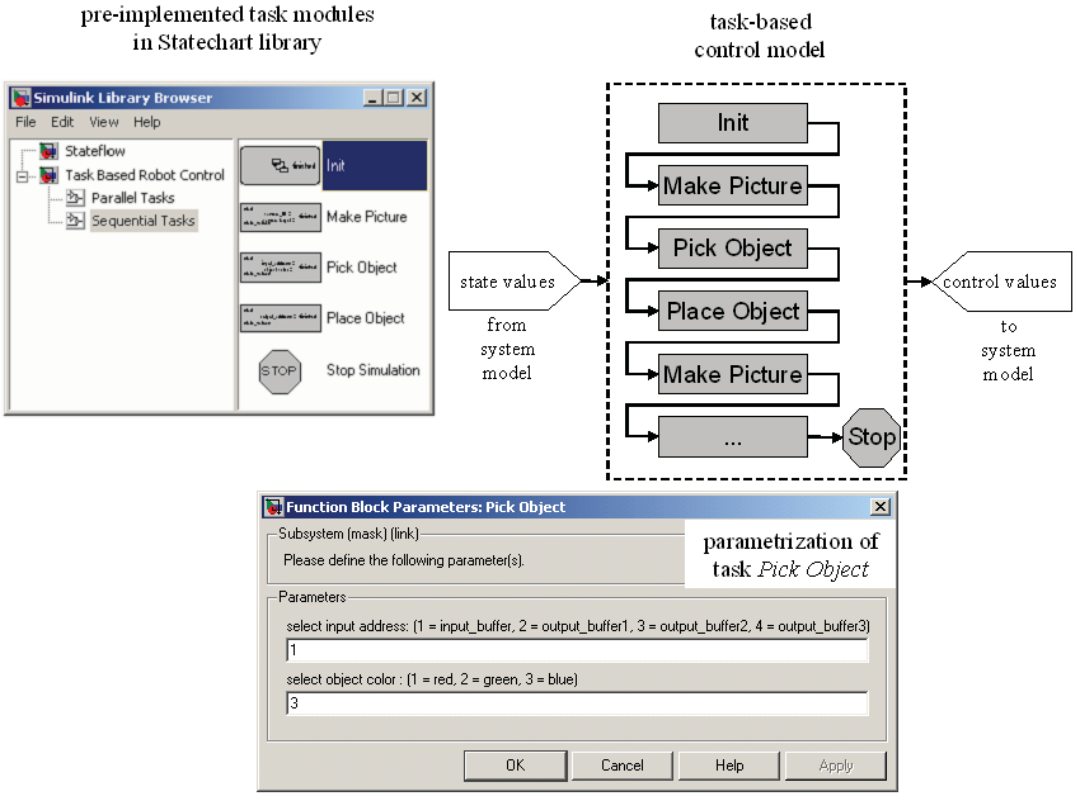

Figure 8. Example of a simple task-based control model specified using pre-defined task modules from library

We call the introduced task-based approach and similar concepts as mentioned in [11] *simple task-based controls*, because the control has to be completely defined on the control model level. Of course, the approach supports much more degrees of freedom in control as traditional machine oriented process controls. However, it may be hard to define a complex and adaptive control with changing control parameters and changing control tasks on this level.

3 Specification and generation of high flexible task-based controls

3.1 Objectives (Supervisory and adaptive task-based control)

In discrete event dynamic systems two types of control are distinguished [8], namely *process control* and *supervisory control*. This section is focused on supervisory control, where control activities only delimit possible states and the particular next state depends on the next event in the system. On the other hand, it can be necessary to adapt the control structure and its parameters due to the new system state. That means in context with the task-based control concept introduced in Section 2.3 that a supervisory controller *SCONT* defines a set of tasks, $SCONT=\{task_1, task_2, ..., task_i, ..., task_n\}$, and a set of permissible parameters $P=\{p_1, p_2, ..., p_i, ...,p_m\}$ for each $task_i$. During operation the controller adapts its active task set and the task parameters depending on the actual system state. In a flexible control the control goal can be reached using different task sets, i.e. the controller can use a subset $SSCONT \subset SCONT$ of tasks to fulfil the defined goal.

As mentioned in the previous section it may be hard or impossible to define such variability exclusive in the control model layer. It seems advisable for high flexible controls

to use a meta-model to describe admissible task sequences and task parameters and

to adapt the control model depending on the actual system state during operation.

3.2 The System Entity Structure and Model Base framework

Rozenblit and Zeigler [20] introduced the *System Entity Structure (SES)* and *Model Base (MB)* framework as a general system design approach. Since then, it was used for different applications particularly with regard to simulation-based system design. According to [24], a *MB* is an organized library of pre-defined atomic or coupled models and the *SES* is a structural knowledge representation scheme that systematically organizes all possible structures of a system. Moreover, the framework defines a graph *pruning algorithm* to extract a unique system structure, called *Pruned Entity Structure (PES)*, from an SES and a *model generator* algorithm that synthesizes an *executable simulation model (EM)* from PES. The fundamentals of the SES/MB framework are pictured in figure 9.

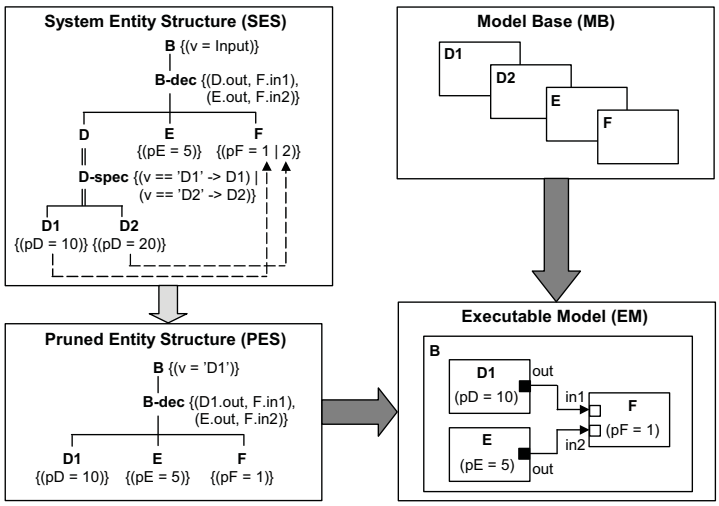

Figure 9. Fundamentals of SES/MB framework, following [24]

An SES describes all variants to compose a modular, hierarchical system and it specifies the various parameterizations. In the following, we summarize important ideas of the SES formalism which are necessary to illustrate the fundamental adaptations for specifying high flexible controls presented in the next subsection.

An SES is a tree, consisting of different kinds of nodes, edges and attached information. The SES formalism differentiates between two general types of nodes: *(i) entity node* and *(ii) aspect node*. An entity node represents a system object. There are two subtypes of entity node in fact *atomic entity* and *composite entity*. An atomic entity cannot be divided into sub-entities. The model base contains a corresponding model for each atomic entity. Composite entities are defined in terms of other entities, which can be of type atomic or composite entity. Thus, the root node *(figure 9: B)* of a tree is always of type composite entity, while all leave nodes *(figure 9: E, F, D1, D2)* are of type atomic entity. The root node and each composite entity node of the tree have at least one successor node of type *aspect node*. That means, there is an alternate mode between entity and aspect nodes. The properties of the different node types can be summarized briefly as follows:

1. *entity node (figure 9: B, E, F,D, D1, D2):* An entity node is defined by a name and is of sub-type atomic or composite. Both types may have attached variables. A composite entity node has at least one successor of type aspect node.

2. *specialization aspect node (figure 9: D-spec):* A specialization aspect node is defined by a name and a set of successor nodes. In the tree it is indicated by a double-line edge. A specialization aspect nodes define the taxonomy of predecessor entity node (figure 9: D) and specifies how the entity can be categorized into specialized entities (figure 9: D1, D2). The specialization aspect node has always successor nodes of type atomic entity. These represent the possible specializations. A specialization aspect node defines selection rules to control the way in which a specialized entity is selected during the pruning process.
3. *decomposition aspect node (figure 9: B-dec):* A decomposition aspect node is defined by a name, a set of successor nodes and coupling information. In the tree it is indicated by a single-line edge. A decomposition aspect node defines a possible decomposition of its parent entity node. It has any number of successors of type atomic and/or composite entity. The coupling specification describes how the sub-entities, represented by the successor nodes, have to be connected. Each coupling information is defined by a 2-tuple. The 2-tuple consists of sub-entity source and destination information. The composite entity *B* in figure 9 has only one-decomposition variant from its viewpoint, consisting of a specialization of entity *D* and the entities *E* and *F*.
4. *multiple-aspect node (not contained in figure 9):* The definition of a multiple-aspect node is quite similar to a decomposition aspect node. However, it defines additionally a *NumberRangeProperty* and has only one successor node of type atomic entity. In the tree it is indicated by a triple-line edge. A multiple aspect node also defines a decomposition of a composite entity, but all sub-entities have to be of the same entity type. Only the number of sub-entities is variable according to the attached *NumberRangeProperty*.

Beside the different node types, the SES formalism supports *selection constraints* for entity nodes, depicted as dottet arrow from an entity node or entity parameter to other entity nodes or entity parameters.

An PES extracted from an SES contains only entity nodes and decomposition aspect nodes. All decisions attached to specialization aspect nodes or multiple-aspect nodes and selection constraints are resolved during the pruning process. Based on the information in the PES the model generator synthesizes an executable simulation model using the pre-defined models in the MB.

3.3 Employing the System Entity Structure and Model Base framework for high flexible controls

Our employment and adaptation of the SES/MB framework for specification, organisation and operation of high flexible task-based controls is based on the development in [24]. In the context of task-based controls *predefined control tasks*, as described in Section 2.3, are deposited in a MB. These tasks appear as atomic entities in an SES and therefore they are always leaf nodes of the graph. In contrast, the *control model* is the most upper composite entity. It represents the root node of a SES. The other composite entities in the graph represent *coupled tasks, task specializations or multiple tasks*. The specific characteristic of composite entities is defined by their subsequent aspect node. In the following, we restrict our description on coupled tasks.

Figure 10 illustrates the fundamental ideas of specifying a flexible task-based control with an SES using a small assembling problem. Similar to the robot control example used in Section 2.3, we assume a robot cell consisting of an input buffer with different units and an output buffer as assembling area. The content of the input buffer is time-variant. The robot is equipped with a camera to identify units in the input buffer. One assembling operation requires following task sequence:

Make Picture: identification of units in the input buffer, selection of one appropriate unit according to the assembly schedule and its position identification (evaluation of coordinates),

Pick Object: robot movement to the selected unit and grabbing the unit,

Place Object: robot movement to the assembling area, adjustment and assembling of the unit.

Moreover, there are two further tasks: *(i) Init* for initialisation of an assembling process and *(ii) Stop* for its termination. The five tasks are retrieved as model components in a MB. The SES in figure 10 shows essential parameters of the basic tasks and specifies the task sequences for the three possible assembling variants:

Bo – La – Ba – Pi1,

Bo – La – Pi1 – Ba,

Bo – Pi1 – La – Ba

A flexible control must have the ability to adapt its control behaviour to the process behaviour. That means for the assembling example, the assembling schedule depends on the available units in the input buffer with the assembling constraints in mind. Hence, it is not possible to extract a complete PES and to generate a control model for the whole assembling process in advance. Therefore, a *new pruning and model generation algorithm* for stepwise graph analysis and model generation is developed. The *first graph analysis* of the SES is marked

with bold edges in figure 10. The *first temporary control model structure* generated from the SES is shown in the upper picture in figure 11. The graph analysis starts at the root node. Then the graph is traversed using a combined breadth-depth approach, controlled by information specified on the aspect nodes. In our example these are the coupling information on the aspect-decomposition nodes. If the graph analysis determines a decision node or decision statement, it stops, derives a partial PES and generates a temporary control model for process control. In our assembling example the task *MakePicture* contains such decision statements, which are characterized by question marks, e.g. *(IdObj=?(BO))*. The first temporary control model, pictured in figure 11, consists of the task sequence: *Init→ MakePicture*. The task *MakePicture* has to identify the first assembling unit *Bo* in the input buffer. It repeats the identification process till it returns with an appropriate object reference or an "error message" after reaching a defined time out. After returning with valid object data, the graph analysis proceeds. Next determined tasks are *PickObject* and *PlaceObject* in the layer of decomposition *Level0-dec* and *MakePicture* in the superior decomposition layer, named *Level1-dec*. Here the second graph analysis stops because of the decision statement *(IdObj=?(La | Pi1))* attached to task *MakePicture*. The structure of the second temporary control model is illustrated in the lower picture of figure 11. The tasks *PickObject* and *PlaceObject* receive the actual object data via their input couplings.

The introduced example of a high flexible control illustrates only a subset of the adapted SES/MB framework. A full syntax and semantics description of the adapted SES formalism for specifying and operating high flexible controls can be found in [16].

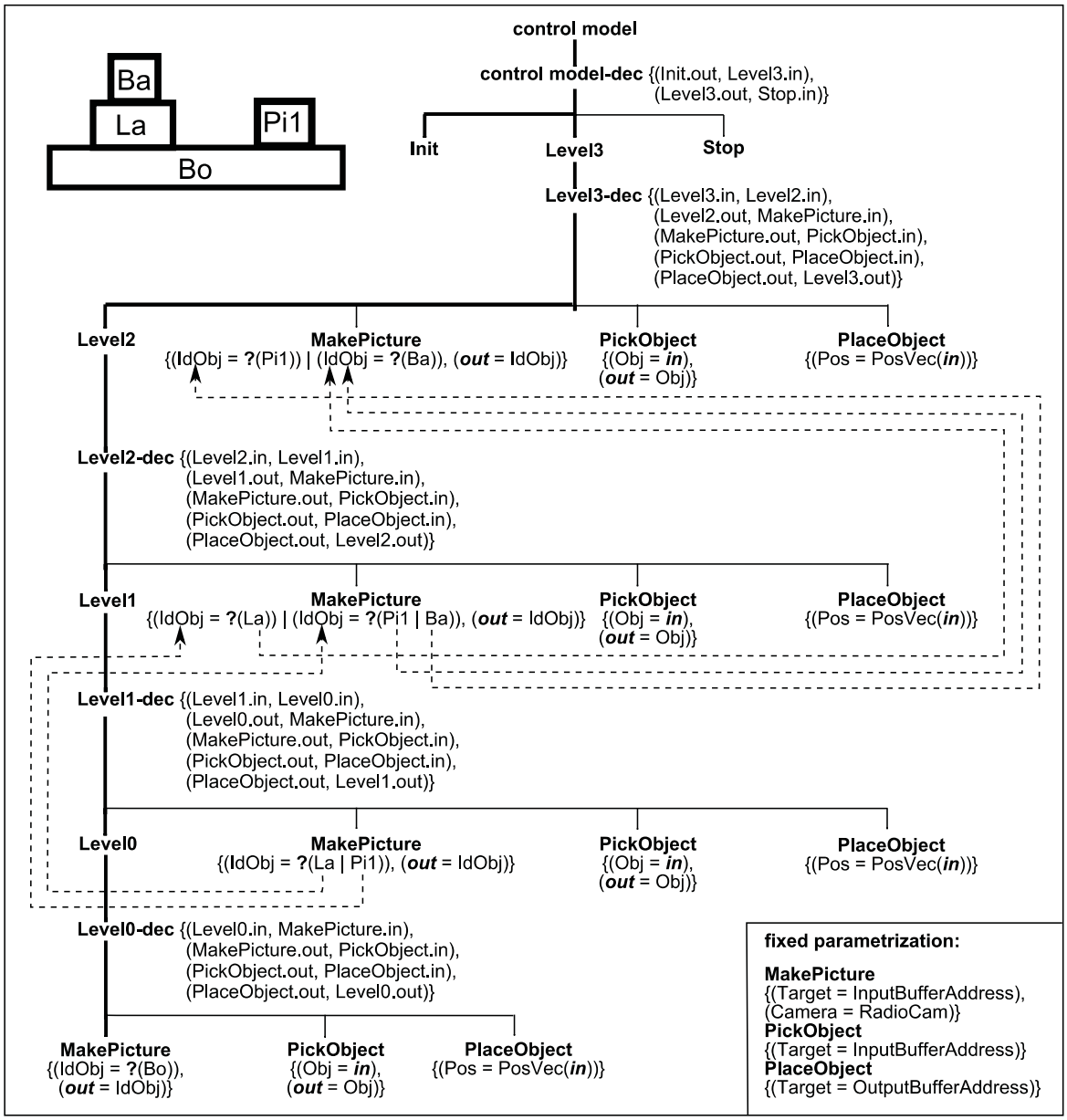

Figure 10. Specifying task-based control variants for a small flexible assembling problem using SES and displaying of the first partial graph analysis (bold edges)

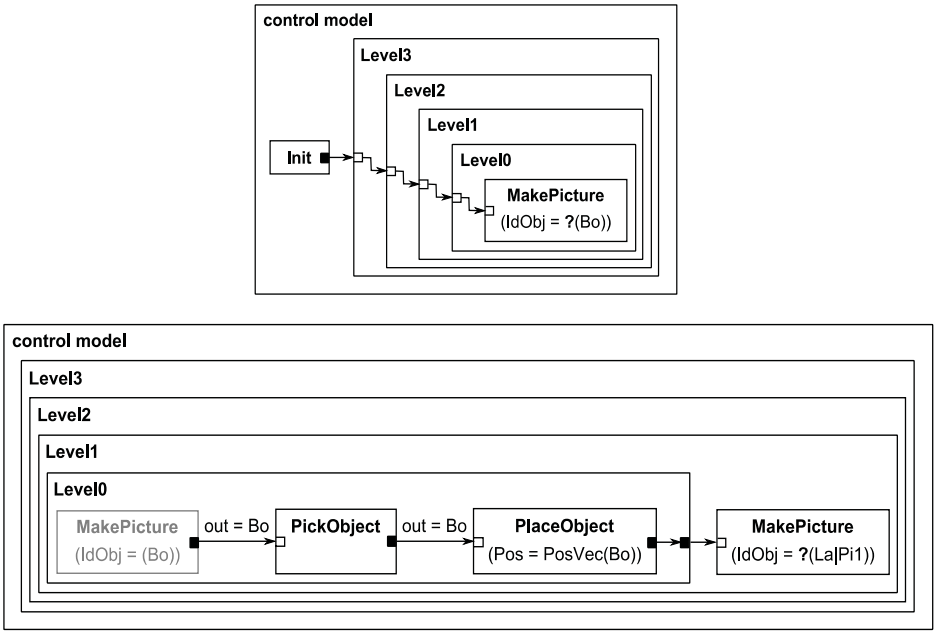

Figure 11. 1st and 2nd control model structure generated from the SES in figure 10

4 Prototyping environment

Test and validation of this research is carried out through a prototype development based on a KR3 robot from KUKA. For robot programming, KUKA offers a special imperative programming language, called KUKA Robot Language (KRL). The KRL is similar to the programming language C, however with less functionality. Concurrent programming features, effective code debugging tools or visualization routines are not available. Moreover, the integration of additional hardware is limited to KUKA products.

In the prototyping environment the robot hardware was extended by adding a PC including the software environment Matlab/Stateflow and further Matlab toolboxes as shown in figure 12. Moreover, a non-KUKA radio camera and an electro-mechanic gripper are mounted on Tool Center Point (TCP) of robot. The structure of multi-level robot process interface for the application presented figure 12 is shown in figure 13. Goal of application is to identify coloured cubes in an input buffer and to sort them by colour in different output buffers.

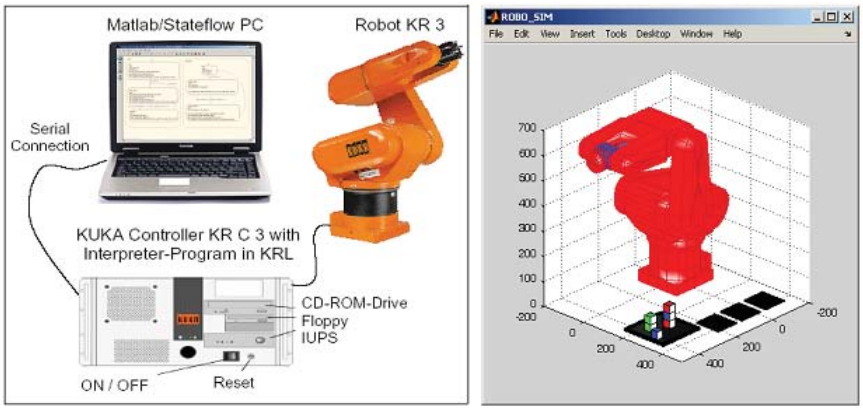

Figure 12. The prototyping environment and visualization of a simple robotic application

The complete control program is running on the PC and it has to be implemented in conformity with the introduced simulation model based control approach using Matlab tools like Stateflow, Simulink and SimEvents. In this application *High Level* and *Low Level* part of robot system component save state information of physical robot and the additional radio camera. The *system model component* of robot sends state values to the *control model* part of simulation model based control system and receives control values from there. Moreover, the *Low Level* part receives information about the physical process devices in Matlab variables and sends instructions to the physical robot and radio camera via blocking or no blocking Matlab function calls. The translation from Matlab code to binary code for the KUKA controller and the radio camera and reverse is implemented in two *additional Matlab toolboxes*, running on the PC, too. Aside from that, the toolboxes provide much more functionality. For example, the complete image processing is done using functions of the Matlab Image Processing toolbox. A further element of process interface is the *conventional KUKA controller*. It is connected with the PC

(serial interface, USB or Ethernet) and works as execution controller of robot. The *interpreter on the KUKA controller* is implemented in KRL and realizes a bi-directional communication with the PC. The interpreter is responsible for the identification and execution of all robot oriented instructions that are transmitted by the PC. Hence, the original KUKA controller operates as execution controller for the robot. Its powerful functions, like *(i)* perpetuation of security given by KRL through workspace supervision, *(ii)* checking final position switches of every robot axis etc., are preserved.

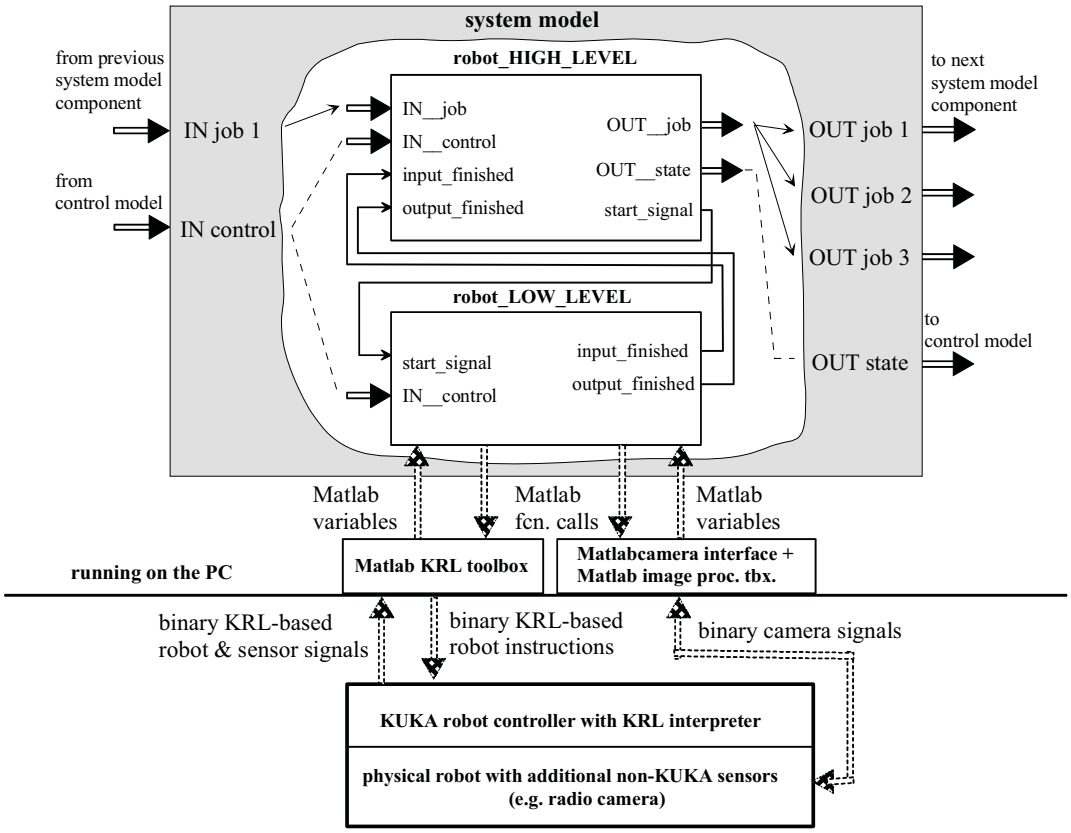

Figure 13. Example of a multi-level process interface for a robot application with image processing

The adapted *SES/MB framework* is also running on the PC and is strongly coupled with the simulation model based process control system. It supports:

the organization of pre-implemented task-based control components in a statechart library,

an SES based specification of high flexible control strategies using XML and

a re-configuration of the control model at runtime.

The graph analyzer (pruning algorithm), the model generator and the bi-directional communication between the simulation model based control system and the SES/MB framework is implemented with Matlab functions, which are bundled up to a Matlab SES toolbox [6, 16]. First practical experiments with the prototyping environment showed that the SES/MB framework and the simulation model based process control system can be performed on an ordinary PC under real time conditions.

To facilitate the development and test of simulation model based controls according to the rapid control prototyping approach [1], a *visualisation toolbox* was created. Thus, it is possible to visualize a robot application including all movement actions. Figure 12 exemplifies a visualization of a simple robotic application. The robot control commands available in the visualization environment have the same semantics as the ordinary KRL instructions transmitted to the physical robot. Thereby, control programs can be developed and tested independently from hardware for different process scenarios. A more detailed description of prototyping environment is published in [13].

5 Conclusions

The introduced approach shows, that it is possible to stepwise extend discrete event-oriented simulation models from the *Design Phases* to a model-based *Process Control System*. An important aspect of the work has been identifying the need for an appropriate modular structure to separate process dynamics, control logic and process interfaces. The result is a simulation model with two fundamental components, called system model and control

model. Particularly the system model has to map the physical process devices in appropriate model components strictly. This allows the simple addition of a process interface to the system model components to interact with real physical process devices or with specific process controllers, which operate as execution controllers. In the *Process Control Phase* it is fundamental, that the simulator is synchronised with real time. Control strategies are already implemented and tested during the *Design Phase* can be used without modifications in the *Automation* and *Process Control Phase*. The system model always saves the actual process states. Thus, it allows model-based calculations to save physical sensors and to improve the process control. It has been shown that the approach also supports a real task-based control specification. Moreover, the integration with an adapted general system design framework has been presented. This integration provides comprehensive possibilities to design and to implement high flexible controls. Development of experimental applications has shown that the continuous implementation of the approach with a uniform methodology within a homogeneous software environment is advantageously. Thus, the software development effort could be significantly reduced, not least because of multiple re-use of software components from the early system design phase in its real operation phase.

The research is being extended into the area of optimal control systems using a superior optimization layer that interacts with the process control system [10]. Other ongoing works are the adaptation of new enhancements of the System Entity Structure formalism [25] and the integration of learning ability along with recurring analyses of System Entity Structures [17].

6 References

[1] Abel, D., Bollig, A.: *Rapid Control Prototyping – Methoden und Anwendungen.* Springer-Verlag, 2006 (in German).

[2] Anderson, C.: *Tomorrow's ideas at work today - scalable flexible manufacturing.* Advanced Manufacturing Magazine, Research Report 2000, www.advancedmanufacturing.com/March00/research.htm.

[3] Bajcsy, R.: *Position Statement Robotic Science.* In: Robotics Research - 12th International Symposium ISSR, Springer Tracts in Advanced Robotics, (Ed.: R. Bajcsy), Springer-Verlag, 2007, 583 – 587.

[4] Dilts, D.M., Boyd, N.P., Whorms, H.H.: *The evolution of control architectures for automated manufacturing systems.* Journal of Manufacturing Systems, Vol. 10, 1991/1, 73–93.

[5] Gausemeier, J., Frank, U., Steffen, D.: *Intelligent Systems, Self-optimizing Concepts and Structures.* In: Reconfigurable Manufacturing Systems and Transformable Factories, (Ed.: A.I. Dashchenko) Springer-Verlag, 2006, 719 – 742.

[6] Hagendorf, O., Pawletta, T.: *A Framework for Simulation Based Structure and Parameter Optimization of Discrete Event Systems.* In: "Book project about advances in DEVS" (ed. Wainer, G.), CRC Press, 2009 (submitted 2008/09/14, 25 pages).

[7] Harel, D.: *Statecharts: A visual formalism for complex systems.* Science of Computer Programming, North Holland, Vol. 8, 1987, 231-274.

[8] Hruz, B., Zhou, M.C.: *Modeling and Control of Discrete-event Dynamic Systems.* Springer-Verlag, 2007.

[9] Hu, X.: A simulation-based software development methodology for distributed real-time systems. PhD-Thesis, University of Arizona, 2004.

[10] Kremp, M., Pawletta, T., Colquhoun, G.: *Simulation-Model-Based Process Control of Discontinuous Production Processes.* In: Advances in Manufacturing Technology - XX, 4th Int. Conf. on Manufacturing Research (ICMR06), Liverpool, UK, 2006, 49-54.

[11] Kruth, J.-P., Van Ginderachten, T., Prianggada, I.T., Valckenaers, P.: *The use of finite state machines for task-based machine tool control.* Computers in Industry. Vol. 46, 2001, 247-258.

[12] Maletzki, G., Pawletta, T., Pawletta, S., Lampe, B.: *A model-based robot programming approach in the MATLAB/Simulink environment.* In: Advances in Manufacturing Technology - XX, 4th Int. Conf. on Manufacturing Research (ICMR06), Liverpool, UK, 2006, 377-382.

[13] Maletzki, G., Pawletta, T., Pawletta, S., Dünow, P., Lampe, B.: *Simulationsmodellbasiertes Rapid Prototyping von komplexen Robotersteuerungen (Simulation Model Based Rapid Prototyping of Complex Robot Controls).* atp – Automatisierungstechnische Praxis, Oldenb. Verlag,Vol. 50, 2008/8, 54-59 (in German).

[14] Pels, H.J., Wortmann, J.C., Zwegers, A.J.R.: *Flexibility in manufacturing: An architectural point of view.* Computers in Industry, Vol. 33, 1997, 271–283.

[15] Peters, B.A., Smith, J.S.: *Real-time simulation-based shop floor control.* In Proc. of ArenaSphere '98, Pittsburgh/PA, USA, 1998, 188-194.

[16] Pingel, T., Pawletta, T.: *SES toolbox V0.1 for Matlab/Simulink* . Internal Report, Wismar University of Technology, Business & Design, RG Computational Engineering & Automation, 2008/10.

[17] Pingel, T., Pawletta, T., Pang, P. S., Kasabov, N.: *Design of a Rapid Prototyping Environment for Intelligent Robot Controls.* In: 4th Int. Conf. on Autonomous Robots and Agents (ICARA 2009), Wellington, New Zealand, Feb. 10-12, 2009 (submitted for publication).

[18] Praehofer, H., Jahn, G., Jacak, W., Haider, G.: *Supervising manufacturing system operation by DEVS-based intelligent control.* IEEE Computer Society Press, 1994, 221-228.

[19] Robinson, S.: *Discrete-event simulation: from the pioneers to the present, what next?.* Journal of the Operation Research Society, 56 (2005)6, 619-629.

[20] Rozenblit, J.W., Zeigler, B.P.: *Concepts for knowledge-based system design environments.* In: Proceedings of the 17th conference on Winter simulation, San Francisco, US, 1985, 223-231.

[21] Saanen, Y.A., Verbraeck, A., Rijsenbrij, J.C.: *The application of advanced simulations for the engineering of logistic control systems.* In: Proc. 9th ASIM Dedicated Conference on Simulation in Production and Logistics, eds. K. Mertins, M. Rabe, Berlin, 2000, 217-231.

[22] The MathWorks: *Control Design Products – Model-Based Design for Embedded Applications.* The MathWorks Report, 2000, 1-16.

[23] Zeigler, B., Kim, J.: *Extending the DEVS-scheme knowledge-based simulation environment for real-time event-based control.* IEEE Transactions on Robotics and Automation, 3 (1993), 351-356.

[24] Zeigler, B.P., Praehofer, H., Kim, T.G.: *Theory of Modeling and Simulation*. 2nd Ed., Academic Press, 2000.

[25] Zeigler, B.P., Hammond, P.E.: *Modeling and Simulation-Based Data Engineering.* Academic Press, 2007.