T U M

I N S T I T U T F Ü R I N F O R M A T I K

Evaluation of Petri Net and Automata Based Description Techniques: An Industrial Case Study

Alexander Sabbah, Robert Sandner

TUM–I9923
Dezember 99

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-12-I9923-100/1.-FI
Alle Rechte vorbehalten
Nachdruck auch auszugsweise verboten

©1999

Druck: Institut für Informatik der
Technischen Universität München

Evaluation of Petri Net and Automata Based Description Techniques: An Industrial Case Study[1]

Alexander Sabbah
Institute for Machine Tools and Industrial Management - iwb
Technische Universität München
email: sb@iwb.mw.tu-muenchen.de

Robert Sandner
Department of Computer Science
Technische Universität München
email: sandnerr@informatik.tu-muenchen.de

Abstract. In this case study, three description techniques representing Petri net and automata based approaches are applied on an industrial application: The control system for the tool changer of a manufacturing cell is developed. The techniques to be compared include common extensions both to the basic Petri net and automata concepts. In order to evaluate the applicability of the approaches, a set of practice oriented criteria is presented in the paper. Based on the experiences gained by the development of the control system, the approaches are evaluated to these criteria.

[1] This work originates from the FORSOFT project, supported by the Bayerische Forschungsstiftung.

Contents

1. Introduction 3
2. Criteria for the comparison: 4
   2.1 Methogical requirements 4
   2.2 Expressability of interesting properties 4
   2.3 Tool support 5
3. Informal description of the modelled system 5
4. Comparison of the different models 6
   4.1 Wessa, a petrinet based approch 6
      4.1.1 Background 6
      4.1.2 The model of the control system 8
      4.1.3 Evaluation of the description technique 10
   4.2 The AUTOFOCUS model 11
      4.2.1 Background 11
      4.2.2 The model of the control system 12
      4.2.3 Evaluation of the description technique 14
   4.3 UML based approach 16
      4.3.1 Background 16
      4.3.2 The model of the control system 16
      4.3.3 Evaluation of the description technique 19
5. Conclusion 20
6. References: 21

1. Introduction

Graphical description techniques have raised much attention in the software engineering community in the last couple of years. Partly this trend is based on the success story of object oriented design methods and languages, where graphical representations play a major role. Until a couple of years ago, graphical descriptions where mainly used as ”sketches” for documentation at the very start of the software development process. Thanks to many improvements, the usage of graphical description techniques has been extended to cover the whole software engineering process, from the early stage of requirements specification up to code generation.

Unfortunately, the software engineer today faces the choice between a whole bunch of description techniques for the same specification task. Not only do many communities use their own description technique, also various semantics for the same elements of description techniques are proposed. Among the different approaches, two mainstreams can be identified: Petri net- and automata based approaches. Automata have quite a number of protectionists in the OO-community while Petri nets are widely used in the field of automation.

The intent of this paper is to achieve more clarity concerning the strengths and weaknesses of both concepts from the users point of view. From the theoretical point of view, the properties of both approaches are well known. Opponents to Petri nets complain that they are not modular while adversaries to automata claim that they are not expressive enough. In practice however, both concepts are extended by additional features to overcome those setbacks. For a comparison of both concepts with respect to practical usage, three approaches which involve common extensions of the basic concepts are applied on an industrial example: The control system for the tool changer of a manufacturing cell is specified. Based on the gained experiences, the approaches are evaluated with respect to a set of criteria, which focus on practical usage and are defined in the paper.

The concept of Petri nets is represented by Wessa [1], which is developed at the Institute for Machine Tools and Industrial Management (iwb) at Technische Universität München. To represent the wide variety of automata approaches accurately, two approaches have been included in the case study: First, a UML [2] based approach also developed at the iwb uses specifics of UML Statechart Diagrams. Second, AUTOFOCUS [3], which is developed at the department of computer science at Technische Universität München, defines State Transition Diagrams which aim on a lean semantics to ease understandability and verification.

The paper is organized as follows: Section 2 defines the criteria for the comparison of the approaches. An informal description of the tool changer is given in Section 3. Models for the control system using each of the approaches are presented in Section 4, where also an evaluation with respect to the criteria is given. Finally, the experiences of the comparison are summarized in Section 5.

2. Criteria for the comparison:

In this section, a selection of criteria for the comparison is presented in order to establish a profile of the approaches regarding applicability in practice. Three classes of criteria are considered: First, methological requirements for a description technique include stepwise refinement and modularity. Second, the expressiveness of the approaches is of interest. Finally, basic aspects of tool support are addressed.

2.1 Methogical requirements

- **Modularity:** The question whether a description technique is modular is a frequently used weapon in discussions about the best approach to use. Often though discussions arise on this subject. A common source of these discussions is a different understanding of the term modular. Therefore, the understanding of modularity which underlies the comparison is stated here:

A description technique is modular if properties of a *component* can be established independently of properties of the environment.

- **Reuse:** The reuse of models safes resources in software development. The notion of reuse discussed here involves both the multiple usage of a component and the adaption of a component to meet new requirements.
- **Hierarchical refinement:** Large systems cannot be specified from scratch. Description techniques should support the stepwise development of a component specification by hierarchical refinement. Two criteria regarding refinement are discussed: The offered notations should be sufficient for practical modelling purposes, and refinement should preserve modularity, i.e. properties of an abstract model should also hold after refinement.

2.2 Expressability of interesting properties

- **Control flow:** Both the control flow of a single component and the synchronisation of different components is of interest. Especially the complexity of models for synchronisation is discussed.
- **Time aspects:** Many embedded systems are real time applications where the timing of activities needs to be specified explicitly. The ways of specifying time related properties are examined.
- **Local data:** Both automata and Petri net based approaches are especially suited for the modelling of control flow. Many systems however also need to handle large data structures. The ability of incorporating complex data structures into a specification is investigated.
- **Complexity of models:** An important aspect is the overall complexity of the resulting models using each approach. The complexity of the models of the tool changer is compared focusing on the handling of the model for the software engineer.

- **Active and idle system status:** For the control of mechanical systems, it is important to distinguish whether the system is idle or running. The ability to model these properties shows the flexibility of a description technique.

2.3 Tool support

Both existing tool support for the editing of the graphical specifications and basic aspects of the generation of executable code from these specifications are discussed briefly.

Other, more theoretical criteria as the possibility of underspecification and nondeterminism are ignored, as this case study focuses in particular on a practical comparison.

3. Informal description of the modelled system

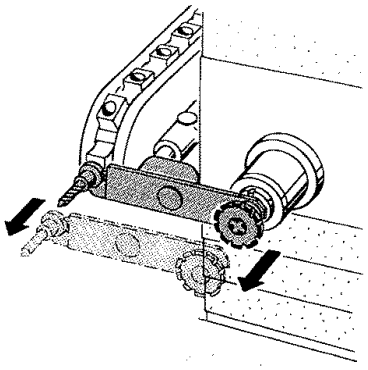

Fig. 1. Tool changer mechanics (Source: Deckel)

The subject of this case study is the control system for the tool changer of the manufacturing cell shown in Fig. 1: To enable the machine to manufacture complex parts, a number of different tools are needed. The active tool resides in the spindle while the temporarily unemployed tools are stored in a chain magazine. The exchange of tools between the magazine and the spindle is performed automatically by a picker arm to save execution time and for safety reasons. Also for safety reasons, the magazine is separated from the workspace by a door.

The control system has to ensure that the tool exchange is performed at minimal time. This includes a maximum of parallelisation of mechanical actions and partly complex low level hardware control. Since for constructive reasons the storage positions of tools in the magazine are not fixed, the system also keeps track of the actual position of the tools.

4. Comparison of the different models

In this section, three different models of the control software using the approaches to be compared are presented. The models adopt a common architecture: The low level hardware control is separated into the control of magazine, door, picker arm and spindle. The coordination task is performed by a flow control unit. The tool change is performed in two phases: In the phase *prepare_change* the magazine performs some preparations, and in the phase *perform_change* the change is actually executed. Due to space limitations, only the flow control and the arm control units are specified in the paper.

The models are discussed in separate subsections which are structured as follows: First, the essential concepts of the description technique are explained. Second, a model of the control system is presented, whereas the flow control and the picker arm control are discussed in detail. Finally, the compliance of the criteria presented in Section 2 is discussed.

4.1 Wessa, a petrinet based approch

4.1.1 Background

Wessa is a flow control with integrated mechanisms for malfunction treatment, which was developed at the Institute for Machine Tools and Industrial Management at the Technische Universität München (iwb) in co-operation with numerous machine tool manufacturers [1]. For the description of the control software a special Petri net based description technique was developed.

Basically all Petri nets used in Wessa are event-condition nets, i.e. that a place can only take up one mark a time. In the here specified notation the specification of an action is done in the transition (Fig. 2, left). An activated transition remains active until an event (acknowledgement) indicates the termination of the action. Thus, the transitions in Wessa are time consumptive. Following the event-condition nets net of *König & Quäck* [4], all transitions also contain an internal condition to enable branching in the Petri net. To fire a transition, all incoming places must be marked and the internal conditions must be fulfilled. Additionally, observation time can be defined in the transition, to indicate the maximum valid duration of an activity.

To enable incremental specification transitions can be refined by sub Petri nets. A sub Petri net must always contain an unique initial and final place. It is considered as an independent net. The interaction between the superior net and the sub net is managed by messages (Fig. 2, middle).

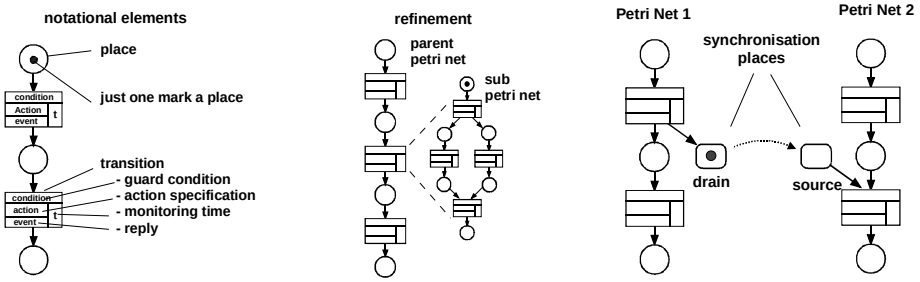

Fig. 2. Syntax constructs in Wessa

To specify synchronisation of two parallel executing Petri nets, Wessa provides similar to the control places of *Moßig & Stäble* [5] so called synchronisation places. The synchronisation is represented by two places distributed on two Petri nets. One place serves as a mark source while the other serves as a mark drain (Fig. 2, right).
Based on the described notation two types of Petri nets are distinguished: modelling and controlling Petri nets (Fig. 3).

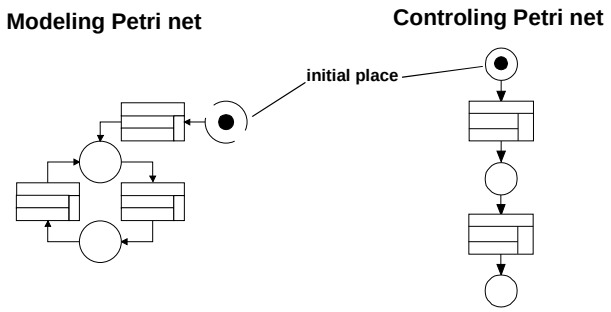

Fig. 3: Modelling and controlling Petri nets in Wessa

Modelling Petri nets serve similar to state diagrams for the description of the dynamic behaviour of software modules. Here the places of the modelling nets represent all the states a software module adopts during its lifetime. The transitions between the places describe the change from one state to the other. Because real objects can only take up one state a time, parallel control flow is not allowed in these nets. In modelling nets all transitions must define the internal condition described above. To fire a transition the incoming places must be marked and its internal condition must be fulfilled. The condition can be changed by messages coming from external Petri nets. After the execution of a transition the outgoing state is marked and maintained until a request for state change is received. Thus, the flow in a modelling net is not continuous.

Controlling Petri nets serve for the description of sequences of control flow. They always contain an unique initial and final place. The initial place describes where the control flow begins, while the final place describes where it ends. In the controlling nets branching, parallelisms and synchronisations are permitted. The control flow is

continuous, i.e. as soon as all source places of a transition are marked, the transition can be fired without regarding an external message.

Modelling and Controlling nets are used to specify Software components in a Client/Server based control software architecture. Each module is specified by one modelling net that describes the outward visible behaviour of the module. The transitions of the modelling Petri net are the services offered by the module. In case of complex services, controlling nets are used specify the flow control during the activation of the transition.

4.1.2 The model of the control system

To clarify the modelling technique used in Wessa the specification of control software for the tool changer mentioned in Section 2 will be discussed here. As described in Section 2 the specification defines the components flow control, door control, arm control, chain magazine control and spindle control (Fig. 4).

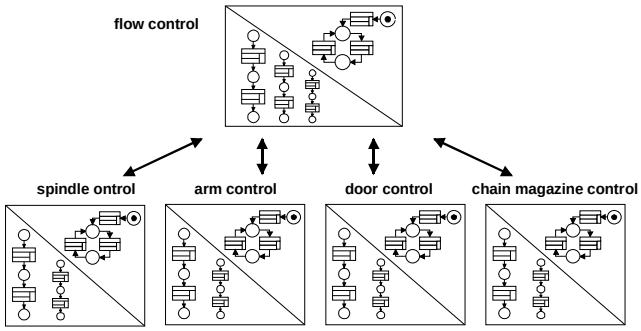

Fig. 4: The five software modules

The components flow control and arm control are to be explicitly detailed (Fig. 5, 6). As mentioned in Section 2 the tool changer offers two main services: preparing a tool change and executing a tool change. These services are specified in the modelling net *pn_flow_control* by two to be refined transitions (Fig. 5, left). The refinement of the transition *execute_change* is detailed in the sub petri net *pn_preforming_change* (fig 5, right). For efficient control the sub net twice specifies parallel execution of transitions. These are the transitions *spindle_gripp chain_magazine_gripp* and *spindle_release chain_magazin_release*. The rest is described as sequential flow. In the transitions of the sub Petri net services offered by other modules are specified. The specification contains the name of the modelling net representing the required module and the wanted service, respectively transitions offered by the modelling net.

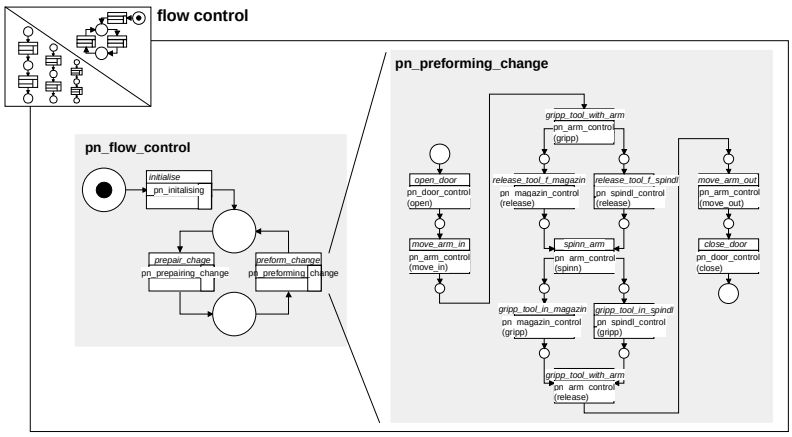

Fig. 5: The module "flow control"

Fig. 6 shows the module "arm control" and its modelling net *pn_arm_control*. In the modelling net, the services offered by the arm control are specified. Further refinement of the transitions *gripp*, *release* or *move_in* is not necessary. In these transitions the specification of hardware connection is done describing the used actors and sensors of the tool changer. Because of the complexity of the transition *spin* it is refined into a sub Petri net called *pn_spinning_tool* (Fig. 6 left). In the transitions of the sub net the remaining specification of hardware connection is done.

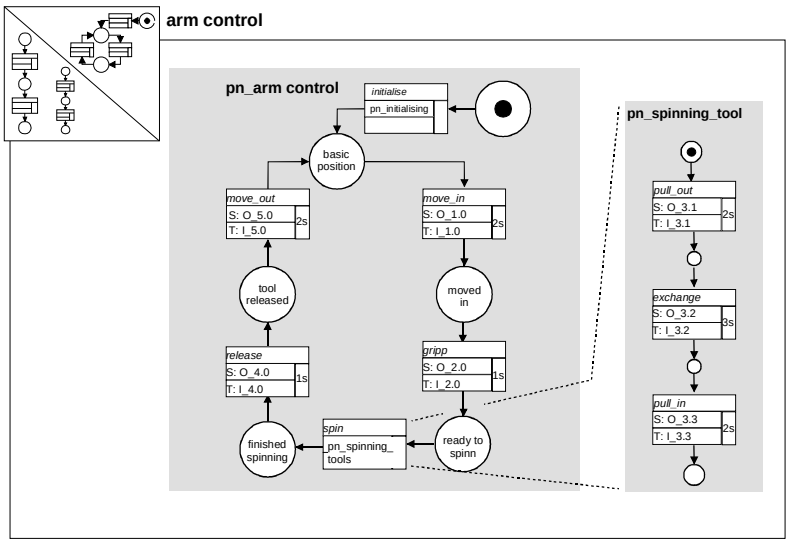

Fig. 6: The module "arm control"

4.1.3 Evaluation of the description technique

The evaluation of the description technique leads to the following results:

Methological requirements

- **Modularity**: According to the definition of modularity described in Section 2, the Wessa description technique is modular. By using the modelling nets, the specification of the behaviour of a component can be established independently of the environment it is used in.
- **Reuse**: This characteristic supports the reusability of modules to a great extent. Due to the fact that all Petri nets are independent units interacting by asynchronous message passing reusability is increased. However, limitations exist when it comes to interactions with new partners. The adoption to a new environment in the sense of new client or server modules is done in the transitions.
- **Hierarchical refinement:** Wessa supports the refinement of transitions through sub Petri nets. This refinement is based on sequential composition. When specifying a complex change of state, a controlling Petri net is used to give detailed description of states and control flow. These sub Petri nets are independent units and can contain parallel flow. However, unlike the refinement of states in automata based approaches the refinement of transitions is always limited to just one entry point an one exit point.

Expressability of the description technique

- **Control flow:** Due to the fact that Wessa originates form a description technique for flow control, it naturally offers the description of features particularly useful for control flow. These include the specification of parallel execution, branchings and iterations. To specify synchronisation between two parallel executing Petri nets, synchronisation places are introduced as mentioned above.
- **Time aspects**: Wessa distinguishes between waiting time and observation time. All time specifications are relative. The definition of waiting time can only be done in the places of a controlling Petri net. The specification of observation time can take place in the transitions of both Petri nets. This time specifies the maximum allowed time for the duration of an activity.
- **Local data**: Generally the purpose of Petri nets is to describe dynamic behaviour and not data handling. Regardless to that fact, local data can be defined in each Petri net in Wessa. This data is limited to simple data types like binary variables and may be manipulated by arithmetical operations defined in the transitions.
- **Complexity of models**: Generally Petri net based models have the affection of becoming rather big in size due to the fact that Petri nets alternate the definition of places and transitions. This is especially noticed when defining parallel or alternative control flow. On the other hand Wessa defines several elements in one

transition (condition, action, termination event) and therefore reduces the number of elements needed in the model. This leads to a rather transparent model of the system.

- **Active and idle system status**: Wessa distinguishes between "idle" and "running" states. All transitions defined in Wessa are "running" states with $t > 0$. The places in a modelling net describe "idle" states with $t > 0$. The places in controlling nets describe "idle" states with $t = 0$, unless waiting time has been specified.

Tool support

The description technique Wessa is supported by a case tool that enables the graphical specification of software for control systems. Additionally the tool can create controlling instructions for special drivers on a real time platform. These configured drivers are executables to control a manufacturing cell. Thus, a full generation of code for control software is possible with Wessa.

4.2 The AUTOFOCUS model

4.2.1 Background

AUTOFOCUS [3], which is developed at the department of computer science at the Technische Universität München, is a formally founded case tool designed in particular for the development of embedded systems and incorporates support for testing and verification. AUTOFOCUS combines several views for the specification of a software system: The structure of a system and its interfaces, i.e. its components and communication relations, are specified using System Structure Diagrams (SSD's). The behaviour of the components is specified in State Transition Diagrams (STD's) which are subject of this case study. Furthermore, scenarios for the interaction of the components are expressed in Extended Event Traces (EET's), a variant of Message Sequence Charts. Finally, types of messages and local data are defined in Data Type Definitions (DTD's) which use the powerful concepts of the functional programming language Gofer [6]. These diagrams support the stepwise refinement of a specification: A component can be refined by a set of sub components in SSD's and EET's, and a state may be refined by a sub STD. A DTD specifying a message type may be refined by a more fine grain specification to refine the interface of a component.

The essential operational concepts of AUTOFOCUS State Transition Diagrams are explained below. Components interact by sending and receiving messages over channels specified in an SSD and may further incorporate local variables which are specified in Gofer syntax as mentioned above. Local variables are quite useful to reduce the number of control states in a specification by partitioning the system state into a visualised control state and a data state.

States are places in control flow, in which a system may reside for a duration longer than zero. Thus states may both express a stable status of a system or a place in control flow which is subject to refinement by further actions. To model the latter, a state may

be refined by a sub STD. No actions are attached to states directly. The refinement of states is explained in detail along with its application in the next subsectcion.

Transitions, which express the activity of a component, are an atomic concept in STD's, i.e. they can be neither interrupted nor refined in STD's. A transition depends on a condition to be taken which may consider the receipt of messages and the data state of the component. Actions attached to a transition include the sending of new messages and the update of the data state. The latter is expressed in Gofer syntax and may involve complex operations. The conditions and actions are seperated by colons:

```
PreCond(data_st):Rec_Msg?Ch:Sent_Msg!Ch:PostCond(data_st)
```

4.2.2 The model of the control system

At the start of the specification process using AUTOFOCUS, the structure of the system is specified in an SSD. Since the communication infrastructure is specified here, the SSD of the flow control is shown in Fig. 7. It structures the system as explained in the introduction of Section 4. The behaviour specification of the flow control and the picker arm, which is specified using STD's, is discussed in detail below.

The behaviour specification of the flow control starts with the STD shown in Fig. 8 which specifies the interaction with the environment. After an initialisation phase, the system can subsequently perform the services *Prepare_change* and *Perform change*. The messages for the communication are specified in a DTD using Gofer syntax:

```
data Input_type =
     Init | Prepare_change(Int) | Perform_change
```

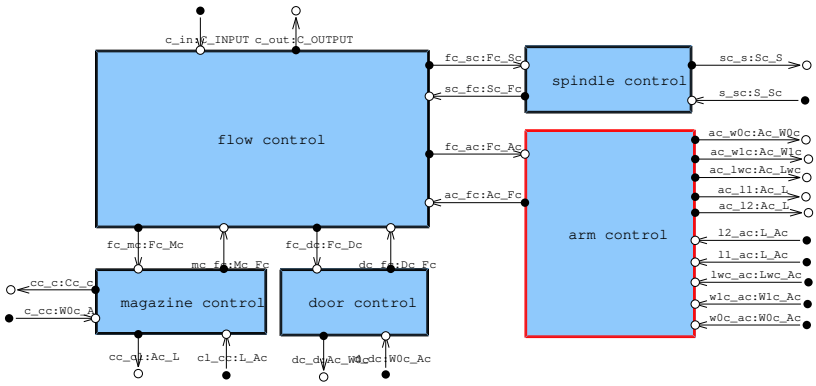

Fig. 7: The static structure of the control system

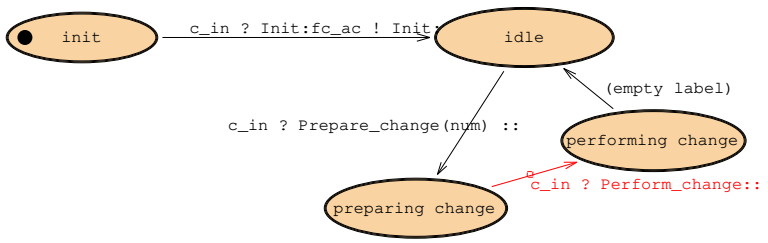

Fig. 8: The top level STD for the flow control

Since transitions are atomic, the execution of the services which last some time have to be modelled by states. In AUTOFOCUS hierarchical refinement of states is used for the detailed specification.

The refinement of the state *performing change* is shown in Fig. 9. To link the transitions to this state shown in Fig. 8 to the transitions in the sub STD, the concept of connectors is used in AUTOFOCUS: For each transition to/ from the state *performing_change*, an entry/ exit point is inserted and linked to transition in the sub STD. These connectors enable to consider the context in which the sub STD has been entered. The conditions for entering and leaving the sub STD specified on both levels are combined by conjunction.

As mentioned before, the parallel execution of mechanical actions is necessary for efficiency reasons. For example, in the state *sp./m. rel. tool* the release of gripped tools by the spindle and the magazine is awaited. To model the synchronisation in a compact manner, a local variable consisting of a pair of Boolean values is used. For both the spindle and the magazine releasing its tool, a looping transitions setting one of the values to False is specified. The execution proceeds as both values are set to False.

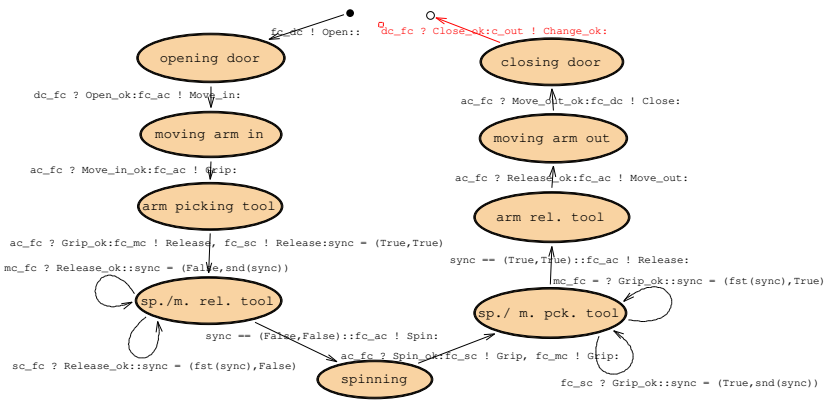

Fig. 9: Refinement of performing change

The arm control, which is specified in Fig. 10, performs the low level interaction with the mechanics of the picker arm and keeps track of the status of the mechanics. Therefore the possible state changes performed by the mechanics are modelled in the specification. In order to save space, the specification is given using a single STD. The messages *Move_in*, *Grip*, *Release* and *Move_out* are translated to appropriate messages for the interaction with the hardware. The messages are defined using DTD's as shown for the flow control. The execution of the *Spin* operation is performed by an appropriate sequence of hardware interactions.

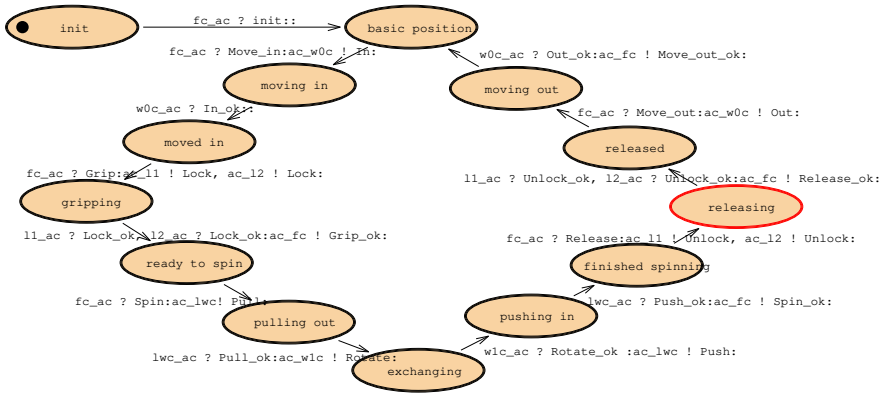

Fig. 10. The arm control STD

4.2.3 Evaluation of the description technique

The evaluation of the model according to the criteria described in Section 2 leads to the following observations:

Methological requirements:

- **Modularity:** AUTOFOCUS specifications are clearly modular with respect to the definition given in Section 2. All kind of behaviour visible at the interface of a component - its ports - can be established independently of the environment in witch the component is used.
- **Reuse:** Due to the concept of ports, components can be reused in different environments, if the same messages are used for communication in each case. If these messages are subject of change however, changes to the whole STD may be necessary as the messages are used in the transitions. This is a common property of both Petri net and automata approaches.
- **Hierarchical refinement:** The state refinement concept explained above of AUTOFOCUS is quite adequate for refinement in terms of a more detailed specification of actions, as used above. The facility of connectors enables the consideration of the context in which a sub STD is executed. Further it keeps different levels of abstraction in a specification independent from each other and thus modular. The state refinement concept use in AUTOFOCUS is especially powerful in combination with combination with structural refinement of SSD's and interface refinement of DTD's.

Expressability of interesting properties:

- **Control flow:** As commonly known, automata are in general well suited for the visualisation of sequential control flow. Due to the concept of state variables, also synchronisation of parallel control flows can be modelled compactly, although not in a visual manner like in Petri nets.
- **Time aspects:** AUTOFOCUS does not provide a particular notation to express time constraints for a transition. Time related properties were not necessary in the specification described above. Where time constraints are needed, they can be easily modelled by communication with a special timer component.
- **Local data:** AUTOFOCUS allows to specify local data of components using the functional programming language Gofer. This allows a powerful combination of data oriented and control flow orientated specifications. This was especially useful for the specification of the tool management aspect in the magazine control. As mentioned in Section 3, this cannot be discussed in detail here due to space limitations.
- **Complexity of models:** The complexity of the resulting models is moderate as can be seen from the above. The number of graphical elements in a specification is comparatively low due to the powerful notation for conditions of transitions. The number of graphical elements could even be reduced further, but this would be on the cost of complex operational specifications of post conditions in transitions.
- **Active and idle system status:** AUTOFOCUS has no notation to distinguish between idle and active states. The syntax could be easily extended for an appropriate marking of states, though this does not imply a semantics of such a marking.

Tool support:
AUTOFOCUS provides a complete tool environment including graphical editors for the described system views and facilities for the simulation of specifications. The experience with the simulation shows that reasonably efficient code generation for STD's is possible. Code generation can be performed fully automatic for executable systems. Further, facilities for testing of non deterministic systems are offered.

4.3 UML based approach

4.3.1 Background

The third approach in this case study is a description technique based on the Unified Modelling Language (UML) version 1.1 [2]. The UML a is description technique defined by the OMG (Object Management Group). It primarily contains a uniform notation and a meta model. UML is mainly a summary of the methods OOD [7], OMT [8] and OOSE [9], but it is also influenced by other methods like the State Charts by Harel [10].

For the object-oriented modelling of software the UML offers numerous diagrams. These diagrams can specify structural, behavioural and implementation aspects of software modules. In this case study the main focus are Statechart Diagrams to specify the internal behaviour of software components.

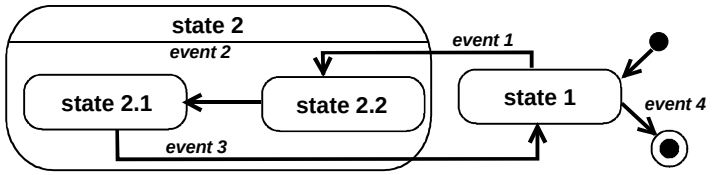

Fig. 11: Basic elements of the Statecharts defined in UML

The UML Statechart Diagram consists of states and transitions as shown in Fig. 11. In UML a state can contain three different kinds of actions. These are the "entry", "exit" and "do" action. The "entry" and "exit" actions are only executed once when entering respectively leaving a state. The "do" action is repeated continuously until the state is left. Additionally, the states defined in the UML can be refined into sub state diagrams which can either have sequential or concurrent running states. The transitions that indicate the connections between the states can be assigned triggering events and optional atomic operations. In UML the following classes of triggering events are distinguished: "signal" events, "call" events, designated conditions and passage of a designated period of time after a designated event.

4.3.2 The model of the control system

To model the control software for the tool changer specified in Section 2 the software is structured into the 5 components: flow control, door control, chain magazine

control, spindle control and arm control. In Fig. 12 these components are shown by using the UML notation for components.

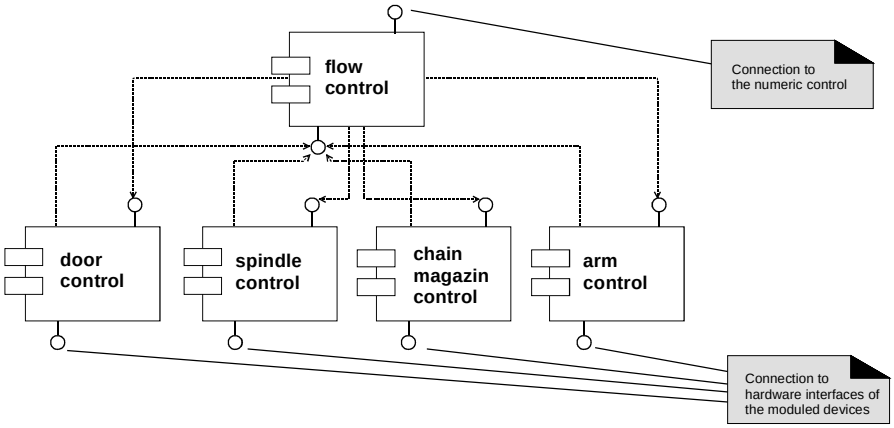

Fig. 12: Structural units of the control software

The interface of the components is represent by circles attached to solid lines. The dashed arrows are used to specify the associations between the components and show that components use services offered by other components.

To specify the behaviour of these software units hierarchical Statecharts as defined in the UML are being used. In the following Section the behaviour for the flow control and the arm control will be detailed. To simplify this process a uniform scheme shall be used, which can be applied to all five control units.

Generally, the scheme defines two super states called *idle* and *running*. These states have to be refined into several sub states. The super state *idle* is refined into all the states the modelled unit has while not executing one of its offered services. The super state *running* is refined into all the states the unit takes on while executing an offered service. To activate the services "call" events have to be received from external units. These "call" events are specified at the outgoing transitions of the sub *idle* states. To certify the correct termination of a requested service, a reply operation is defined in the entry action of the super state *idle*. If the execution of an offered service requires interaction with external units the sub state may be refined into further sub states to encapsulate the interactions with these server units. Though the UML offers the specification of "do" actions in states, no such actions shall be used here, i.e. only "exit" and "entry" actions are specified.

Fig. 13 shows how the Statecharts of UML can be used in conjunction with the specified scheme to model the dynamic behaviour of the flow control. Here the two states "initialised" and "prepared" are sub states of the super state *idle*. The two services of the tool changer *prepar_change* and *perform_change* are specified by the two states *preparing_change* and *preforming_change*. To initiate these services the two events *e_ prepar_change* and *e_perform_change* are specified.

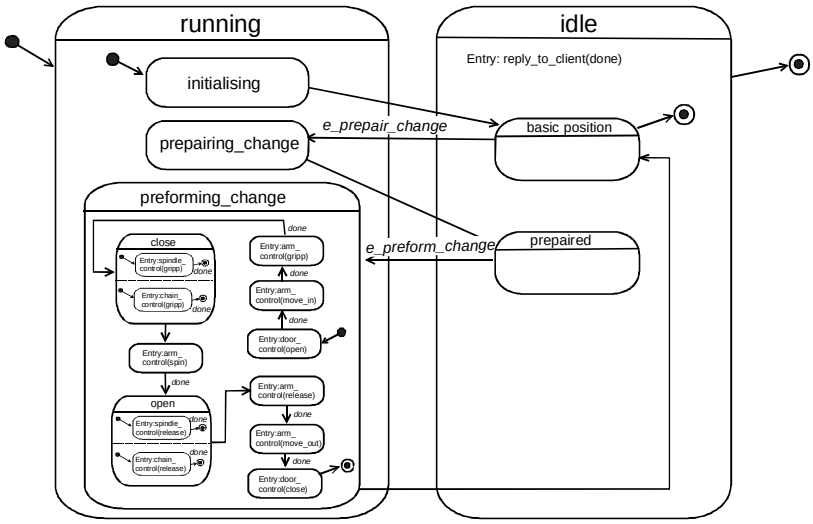

Fig. 13: Behaviour of the flow control

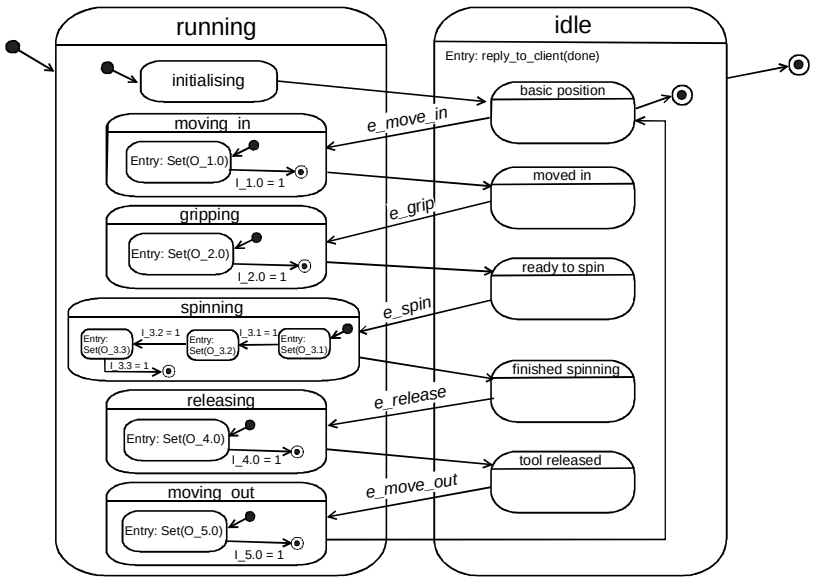

Fig. 4: Behaviour of the arm control

Due to the fact, that the state *preforming_change* defines a complex behaviour that requires several interactions with the other control units, the state is refined into several further states. In the entry action of these states further requests at the other control units are specified. To enable parallel interaction with the spindle control and the chain control the states *grip* and *release* are again refined into concurrent sub states.

In Fig. 14 the behaviour of the arm control is shown, where the scheme introduced above is applied as well. In the sub states of the super state *running* the interaction with the hardware devices in term of actors and sensors are specified.

4.3.3 Evaluation of the description technique

The evaluation of the UML based approach with respect to the criteria introduced in Section 2 leads to the following conclusion:

Methological requirements

- **Modularity**: According to the definition of modularity as defined in Section 2 the UML based approach is clearly modular. The modularity is based on the use of independent classes (active classes) with indivdual interfaces and behaviour. In UML the interface of a component can be specified in a separate interface class that only declares the services the component offers.
- **Reuse:** Due to the fact that the described software is based on a Client/Server architecture the following can be observed: As long as the changes in the environment do not influence the interactions between a component and its server units reusability is supported to a high degree. If these interactions are influenced the changes will have to be considered in the states of the Statechart Diagrams.
- **Hierarchical refinement:** Statecharts in UML supports the hierarchical refinement of states into sub statecharts. This feature is essential for the scheme described above to specify the behaviour of a component. Additionally to the described characteristic transitions can be connected to so called "stubs" that specify connectors to states in a sub Statechart. These sub states are other states than initial or final states.

Expressability of interesting properties:

- **Control flow:** As already described in the AUTOFOCUS Section Statecharts are well suited for the description of control flow. Due to the face that states in UML can be refined sub statecharts containing sequential or concurrent running states, the synchronisation of parallel execution can be specified graphically. A graphical description of synchronisation in the sense of semaphores is not supported.
- **Time aspects**: In UML time aspects can be defined in Statecharts by using *time events* in transitions. This event specifies that a designated period of time after a designated event has to pass before the transition fires. The usage of the key word "**after**" denotes the period of time since the entry of a current state.
- **Local data**: Unlike UML version 1.0 where local data in terms of state variables could be specified in an Statechart Diagram, UML version 1.1 suppresses the specification of local data. Only so called "guard-conditions" in the from of defined variables can optionally be specified in transitions.
- **Complexity of models**: Compared with basic automata concepts the number of elements needed to specify the software module in this approach has increased slightly. The reason for that lies not in the UML based notation but in the defined

scheme that uses the hierarchical refinement to structure the model. On the other hand the structured scheme increases the transparency of the model. This makes it easier to handle of the model and reduces the complexity for the software engineer.

- **Active and idle system status**: Generally the Statechart Diagrams of UML do not distinguish between *running* and *idle* states. By using the scheme defined above the distinction is introduced

Tool support

Basically a large set of commercial case tools support the UML. Unfortunately these tools differ strongly concerning the number of supported UML diagrams. Due to the fact that Statechart Diagrams are essential to specify dynamic behaviour most of the tools do support these diagrams. Further more the scope of automatic code generation differs strongly. Some tools just support the generation of basic structural elements while others enable full generation of executable code.

5. Conclusion

In this case study, we applied three Petri net and automata based description techniques for behaviour modelling on the development of an industrial application: The Petri net based approach Wessa, State Transition Diagrams in AUTOFOCUS and a Variant of UML State Diagrams. These description techniques involve common extensions to the basic Petri net and automata concepts. With practical experience obtained in the study, we evaluated each description technique with respect to criteria which focus on practical applicability. The observations are summarized below:

All three approaches can be applied in a way that they are modular with respect to the specification of components. The reuse of specifications is possible but limitations exist regarding the adaption of a component to meet new requirements. In all approaches, notations for the stepwise refinement of a specification are provided which are sufficient for the modelling tasks occuring in the study.

The compared approaches are expressive with respect to a number of important properties of the developed system. Wessa enables to visualize synchronisation of parallel control flow in a graphical manner and introduces compact and convenient syntax construct for time related properties. AUTOFOCUS is especially well suited to incorporate complex data definitions. UML extends the ability of automata to visualize parallel control flow and offers a syntax to specify time aspects. Regardless of contrary statements found in the literature, e.g. [11], the specifications described in this study are of similar complexity. The evaluation of the criteria is summarised in table 1.

Due to the different focus of the approaches, the amount of tool support concerning code generation differs widely. However, the existing facilities show that a principal tool support is given for all approaches and that the basic problems concerning this subject have already been solved

Criteria	Wessa	AUTOFOCUS	UML based approach
Modularity	+	+	+
Reuse	+	+	+
Hierarchical refinement	+	+	+
Control flow	+	0	+
Time aspects	+	0	0
Local data	0	+	-
Complexity of models	0	+	+
Active and idle states	+	-	0
Tool support	+	0	+

Table 1. Summary of the comparision

These results of this case study yield to the following conclusions: Commonly known drawbacks of the basic concepts are mostly overcome by extensions introduced in modern approaches. Assertions that blame Petri nets for lack of modularity do not hold for extensions as the communicating nets introduced in Wessa. Also the critics that claim automata are not expressive enough are not valid for extended approaches like State Transition Diagrams of AUTOFOCUS or State Diagrams of the UML. As a consequence, the results show that the differences between Petri net and automata based description techniques are becoming increasingly smaller. Surely will every description technique retain its particular strengths and weaknesses. But it is clear to see, that they are moving closer together towards efficient, component based software development.

6. References:

1. G. Reinhart, A. Sabbah. *Werkzeug zur störungsoleranten Steuerung von Abläufen.* In ZWF 92, Vol. 9, pages 440 - 442. Carl Hanser Verlag, München, 1997.
2. G. Booch, J. Rumbaugh and I. Jacobsen. *The Unified Modeling Language for Object Oriented Software Development, Version 1.1,* 1997.
3. F. Huber, B. Schätz, A. Schmidt and K. Spies. *AUTOFOCUS - A Tool for Distributed Systems Specification.* In Proceedings FTRTFT'96 - Formal Techniques in Real-Time and Fault-Tolerant Systems, pages 467-470. Springer Verlag, LNCS 1135, 1996.
4. R. König, L. Quäck. *Petri-Netze in der Steuerungs- und Digitialtechnik.* R. Oldenburg Verlag, München, 1988.
5. K. Moßig, M. Stäble. *Steuerungssymthese mit kontrollierten Free-Choice Petri-Netzen zur Prozeßbeschreibung.* In Automatisierungstechnik 43, Vol. 11, pg. 506**,** 1995.
6. M. P. Jones. *An Intoduction to Gofer*. 1993.
7. G. Booch. *Object oriented analysis and design with applications, 2.nd ed.* Benjamin/Cummings, Redwood City, 1994.
8. J. Rumbaugh, M. Blaha, W. Permerlani, F. Eddy, W. Lorencen, W. *Object Oriented Modelling and Design.* Prentice-Hall, Englewood Cliffs, 1991.

9. I. Jacobsen, M. Christerson, P. Jonsson, G. Övergaard. *Object-Oriented Software Engineering, A Use Case Driven Approach*. Addison Wesley, Workingham 1992
10. D. Harel. Statecharts: A Visual Formalism for Complex Systems. In Science of Computer Programming 8, pg. 231. 1987
11. L. Libeaut and N. Rakoto-Ravalontsalama. *Modelling and Analysis of a Manufacturing Cell: Petri Net vs. Automata Approach*. Published at the Wesic'98 Conference, Girona, Spain, 1998.

[1] This work originates from the FORSOFT project, supported by the Bayerische Forschungsstiftung.