ulm university universität
uulm

Towards a UML Profile on Formal Semantics for Modeling Multimodal Interactive Systems

Concepts for modeling interactive systems using standard tools of software engineering

Marcel Dausend

Ulmer Informatik-Berichte

Nr. 2011-07
Dezember 2011

ulm university universität
uulm

Towards a UML Profile on Formal Semantics for Modeling Multimodal Interactive Systems

Concepts for modeling interactive systems using standard tools of software engineering

Technical Report

Marcel Dausend

Institute of Software Engineering and Compiler Construction
University of Ulm

Table of contents

Abstract

An inherent general problem is that systems become more complex. This problem is exacerbated by the use of additional interaction concepts like multimodality.
Interactive systems are widespread and often apply advanced interaction concepts to ease use and enhance user experience. Touch interaction and multimodality are concepts that are on the rise and are already offered by many commercial products. For example, user interfaces of the infotainment equipment in current premium cars provide speech interaction, especially to increase operational safety.
A common state-of-the-art approach to master complexity comprising all phases of software development is Model-Driven Development. Modeling raises the level of abstraction by using (mainly graphical) models to bridge the gap between specification and implementation.
The Unified Modeling Language (UML) is the de facto standard modeling language. Although UML has proven itself in practice, it does not support modeling of interactive systems and their user interfaces so far.
We present an approach to extend UML for modeling interactive systems. We emphasize on modeling multimodality by using and extending UML state diagrams, by creating UML compliant extensions based on UML profiles, and by defining formal semantics for our extensions including behavioral aspects by means of Abstract State Machines.
We propose an architecture using separate state diagrams for each modality. These state diagrams are synchronized by a common system model. The suitability, usability, and applicability of our approach is reviewed by means of an expert evaluation.
Our approach enables modeling multimodal interactive systems with one formalism. Thus, it supports an integrated kind of modeling as well as separation of different concerns of multimodality. The provided formal semantics for our UML profile enables automated processing of our models including comprehensive tool support for simulation and code generation.

1 Introduction

Interactive systems are a part of our daily life. An example are mobile phones which rapidly evolve and usually offer more than just telephone functions, like being a personal digital assistant and providing entertainment features.

As a consequence, user interfaces (UIs) of mobile phones have significantly changed. A large number of phones provide multimodal UIs for input and output, e. g. touchscreens in combination with speech recognition and synthesis.

Another example of evolution of current UIs is given by premium cars. New features often aim only to improve the needs of comfort, entertainment, and information [6]. The UI of current cars is extended to include speech recognition and synthesis and furthermore offers new options for haptic interaction. This extension is necessary to fulfill the demands of customers and simultaneously ensure the safety of passengers.

These examples comply with Balzert's statement [1] that users ask for systems providing comfortable and intuitive UIs while desiring more functionality. The complexity of systems and their UIs increases steadily. Thus, development of interactive systems is becoming more complicated.

In this technical report, we provide a modeling approach for multimodal interactive systems, taking into account some hypotheses (cf. Sect. 2.2 based on our current results [42]. We analyze alternatives for Unified Modeling Language (UML) extensions (cf. Sect. 2.3) and provide a UML profile for modeling multimodal interactive systems (cf. Sect. 3 and 4) with a formally defined semantics (cf. Sect. 5).

As a proof of concept, we present results of a case study (cf. Sect. 7.1) for validation before discussing an expert evaluation of our modeling approach (cf. Sect. 7.2). We conclude by outlining general ideas for improvements of our approach and sketch how a foundation for UML profile modeling can be achieved, which could solve the problem of integrating formal semantics of profiles into UML's semantics (cf. Sect. 5.2).

2 Formalisms and Methodology

In this section we motivate our approach (Sect. 2.1), before we outline our view of Model-Driven Development (MDD) (Sect. 2.2). We introduce UML for interactive systems, discuss related approaches (Sect. 2.3), and briefly introduce how to extend UML using profiles (Sect. 2.4).

As our approach incorporates topics of different domains, we provide some links to basic literature which might be useful to understand this report:

- Software Engineering [48]
- The UML Reference Manual [44]
- Foundations of Multimodal Dialogue Systems [51]
- Abstract State Machines [5]

2.1 Software Engineering for Interactive Systems

In software engineering (SE), mutual combinations of *formalisms*, *methods*, and *tools* are provided to ease the development of *applications*. The integration of all of those aspects on a common *theoretical basis* is a crucial issue, since none of them really makes sense without any of the others [42].

Interactive systems like phone navigation systems or in-car entertainment systems also put high demands on SE. Currently, different aspects like the interaction between users and the system, dialogs, multimodality, interfaces, and architecture, can only be described by using an adequate combination of formalisms. Hence, using a set of different formalisms results in some negative consequences: The process is highly dependent on the used formalisms and architecture; Usually, there is no integrated tool support. Proposed solutions focus heavily on the integration of graphical aspects and pay little attention to multimodality, if at all.

A major challenge is to provide a SE-approach for interactive systems that covers all aforementioned aspects and supports all phases of development of interactive systems in a consistent way.

Therefore, *formalisms*, *methods* and *tools* on a common *theoretical basis* have to be provided. MDD is a typical instance of our view of SE, as it comprises all those facets. We describe our view of MDD (cf. Sect 2.2, according to [42]) by identifying the main problems of current MDD approaches, thereof deducing major hypotheses and providing solutions for some of the identified problems.

2.2 MDD for Interactive Systems

A state-of-the-art proposal to SE is to use MDD [42]. It raises the level of abstraction by using (mainly graphical) models instead of natural language specifications and conventional programming languages to bridge the gap between specification and implementation. Therefore, appropriate model transformations or direct generation of code out of models can be seen as an integrated part of a development process.

Although MDD is a step into the right direction, we identified some major difficulties and deducted the following hypotheses as a basis for our MDD approach (for details see [42]): base on *established formalisms* and on *formal semantics*, consider *high quality* of *models* as soon as possible, provide integrated *tool support*, and keep *adaptability to specific domains* in mind, if needed. These hypotheses are mostly motivated by pragmatic considerations — in particular, to build on existing good ideas and approaches rather than to reinvent them.

Our vision of an integrated, tool-supported MDD approach bases on the above mentioned hypotheses. Although this vision has not yet successfully been achieved, we have made substantial progress for some important aspects:

- a semantics foundation [32] for UML using Abstract State Machines (ASMs), which are an established formalism
- support for different kinds of quality assurance including comprehensive tool support [18], in particular extensive, automatic code generation (according to the formal semantics) [19, 20]

2.3 Modeling Interactive Systems with UML

Since modeling Human-Computer Interaction (HCI) has been a topic for many years, there already exist some diverse approaches for modeling HCI systems (*ConcurrTaskTree (CTT)* [43], *UsiXML* [34], *UMLi* [8], *UWE* [27], *XIS* [47], *WISDOM* [37], ...). However, we observe that there is currently no preferred approach, neither for developing nor for modeling HCI.

De Melo [13] gives a comprehensive overview and analyzes relevant approaches regarding appropriateness, expressive power, extendibility for multimodality, universality, distribution, understandability, and tool support. He concludes that UML provides the most benefits.

Some other HCI researchers promote UML for modeling interactive systems, too, e. g. Nunes and Cunha [37], Hennicker and Koch [27], and Silva and Paton [8]. It is commonly agreed that extending UML would be valuable to support HCI modeling.

Recent introductions of UI aspects into UML mostly focus on static aspects like structuring presentation elements of interactive systems, e. g. by using class stereotypes for *WISDOM* [37] or by concentrating on aspects of conceptual, navigation and presentation design, e.g as *UWE* [27]. Both approaches use UML profiles to model tasks and to describe graphical elements of UIs, but do not take into account multimodality. Nóbrega et al. [36] deal with behavioral aspects and demonstrate that UML provides the same expressiveness as *CTT* [43] by mapping its concepts to UML activities.

We are not aware of any UML-based HCI approach defining a formal semantics for its behavioral models. Since long, it has commonly been agreed that formal semantics is required and indispensable for automatic processing of models [41], e. g. for simulation or code generation.

In particular, UML state diagrams, which are an extension of statecharts [26], provide the basis for our multimodal HCI extension. An important argument for this decision is the characteristic behavior of UIs which is perfectly reflected by state diagrams: A dialog between the user and a system consists of several steps between dialog states. A dialog state holds until a user or system interaction occurs. Analogously, a transition originating from a UML state only occurs if an event triggers any of the outgoing transitions of that state. Both, dialogs as well as state diagrams, can be seen as reactive systems.

Some concepts of state diagrams can be directly used to model some aspects of interactive applications: Transitions are branched or merged using *junctions*

or *decisions*. This enables modeling alternatives within a dialog or joining different paths of a dialog. *Histories* facilitate to restore previously interrupted dialogs: *Sub states*, which were active at the time of exiting a state, are reactivated on re-entering this state over its *history* pseudo state. *Entry points* and *exit points* define an interface of a state, e. g. to define dialog modules. *Forks* and *joins* can be used to enter or exit orthogonal states, i. e. states with multiple regions. They enable to model different variants to initialize or end a sub dialog.

The appropriateness of approaches for modeling speech dialogs [29] and multimodal dialogs [23], [46] with state diagrams has already been demonstrated. They differ in several aspects of our approach [10], which introduces some extensions.

Kölzer [29, 30] adapt statecharts [26] to model speech dialogs with changing initiative. Both, notation and semantics of statecharts, are substantially redefined to match the requirements of modeling speech dialogs. This redefinition means that the resulting behavior significantly depends on the dialog system processing the model.

Goronzy et al. [22, 23] propose a concept to model multimodal interaction using UML state diagrams. The implementation of the concept is not consistently UML compliant. In consequence, some advantages arising from the use of UML are lost, for example the exchange of models with other UML tools is difficult. The introduced concepts are not integrated into UML as UML extensions and their semantics is implicitly given by the implemented tool. As a consequence, even UML experts have to learn the concepts and their semantics depending on the provided tool.

Modeling multimodal dialogs is restricted in a way, that speech dialogs and graphic-haptics dialogs have to be strictly separated in different state diagrams. The supported formalisms are predefined, so that it is not possible to adapt or exchange formalisms with respect to the target system. For instance, a fixed set of widgets – elements of a graphical user interface (GUI) – implemented in Java is provided by the tool. Similar to our approach, grammars can be dynamically derived from states to sub states. Dynamic composition of graphical representations depending on the state hierarchy is not supported.

Task modeling is preceded by modeling the multimodal interactive application. Therefore, UML can be used for modeling tasks as well, as pointed out by de Melo [13]. He argues that a hierarchical task analysis as with CTT is still possible. Moreover, using a combination of different UML diagrams (e. g. *use case*, *activity*, and *state machines*) may enable stronger cohesiveness of the task model and the dialog model.

Some approaches define UML profiles with stereotypes for concrete widgets to model graphical representations (similar to Hennicker and Koch [28], Martins and Silva [35]). However, such an approach has some drawbacks. The graphical representation is limited to elements which are part of the profile and its stereotypes, e. g. combo boxes or text fields.

The concepts of our approach are specialization of the formalisms chosen for describing the modality specific contents. Depending on the chosen formalism, a widget can be implemented with its individual level of abstraction and interface, e. g. a clock widget containing the behavior of the clock. Using stereotypes for describing widgets is just one example for an appropriate formalism.

2.4 Extending UML

In principle there are two options for extending UML: *Extending the metamodel* or *using profiles and stereotypes*.

Metamodel extension by using inheritance and adding new classes in fact means to create a new language.

In contrast to this, "A *profile* defines limited extensions to a reference metamodel with the purpose of adapting the metamodel to a specific platform or domain." [38, p. 184]. Therefore, our interest with respect to domain-specific languages (DSLs) is focused on extension by UML profiles.

Pardillo [41] gives a systematic review of research papers describing approaches which use UML profiles. It includes 63 approaches from the most relevant conferences in the 11-year period of 1999–2009. Amongst these are UI related approaches like the aforementioned Nunes and Cunha [37], Hennicker and Koch [27], and Silva and Paton [8]. The *abstract definition of profiles* and *quality of presentation* of 39 of these approaches are analyzed considering 26 variables in seven categories. Pardillo [41] states that "the definition of the profile formal semantics is very rare", and reasons that "this pattern also seems natural since UML itself has no formal semantics for their meta classes (which are described in natural language), whereas behavioral modeling needs to be supported by their formal semantics to be useful".

According to our hypotheses (cf. Sect. 2.1) we decided to use profiles for extensions to overcome the aforementioned limitations and problems of existing approaches. Thereby, we investigate how a formal semantics for extensions can be defined and seamlessly integrated into an existing UML semantics (cf. Sect. 5).

In the following, we give a brief technical introduction to UML profiles. A profile coexists beside UML itself and is therefore an independent extension that can dynamically be replaced by other profiles or even reused for a different application [39, 38]. The semantics of a profile is not allowed to be in conflict with UML semantics. A profile can group the following elements:

- Stereotypes (define an extension referring to an element of the metamodel)
- Extensions (enable different kinds of relations between elements, e. g. stereotypes and metamodel elements)
- Constraints (define formal constraints on elements of a profile)
- Textual annotations (describe the semantics of a profile and contain further annotations as prose text)

Fuentes-Fernández and Vallecillo-Moreno [17] describe how UML profiles can be created step by step. However, the topic of how to define a profile's semantics is not covered.

3 Extending UML for Multimodality

As a basis for modeling multimodal interactive systems using state machines, we introduce a UML profile and define stereotypes for speech dialogs as well as graphic-haptic dialogs.

We analyzed common use cases (UCs) of HCI and derived relevant problems for modeling multimodality, e. g. synchronization during multimodal dialogs. Thus, we identified static and dynamic aspects for defining our UML extension.

A concrete UC, which is used for our expert evaluation (cf. Sect. 7.2), is unlocking a mobile phone while driving a car (cf. Sect. 6.1). Further UCs are extensions of current in-car dialogs to introduce new concepts, e. g. multimodal or barge-in functions. The used interaction concepts affect different multimodal elements: speech dialog, haptic interaction, and graphical representation.

3.1 Defining the UML profile

In this section, we define our UI profile consisting of the stereotypes ≪GUI≫, ≪Grammar≫, and ≪Prompt≫ used to extend UML state machines and its notation with static as well as dynamic aspects.

Static aspects are related to the UI description, e. g. a representation of the UI at a certain time, a specific utterance for speech synthesis, and a grammar for speech recognition.

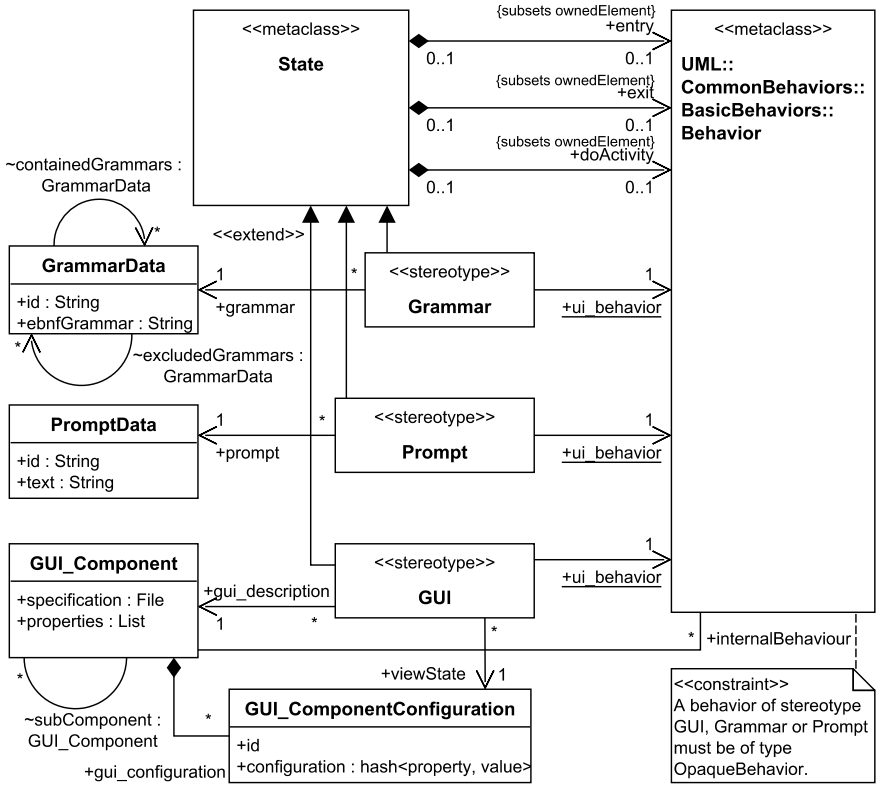
Dynamic (behavioral) aspects concern the dialog between a user and the system. They are defined as changing static aspects over time as a consequence of evaluating a user (or system) action in its temporal and situational context.

The stereotype ≪GUI≫ enables adding widgets (cf. sect. 3.1) to states and computing current screen representations. The stereotype ≪Grammar≫ is used to support speech recognition and the stereotype ≪Prompt≫ implements system's speech output. All those stereotypes are applied by adding one to a state and assigning values to its related properties.

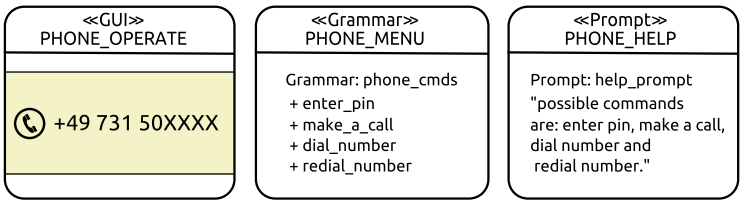
Static Aspects

The static aspects of the stereotypes enable modeling of graphic-haptic as well as speech dialogs (see Fig. 1(a)). Each stereotype extends the class *State* of the UML metamodel. Their properties and associations reference modality–specific contents of each stereotype, e. g. grammar and behavior.

Using state diagrams to describe interaction is very common (cf. Sect. 2.3) and as we think intuitive, because a state symbolizes an invariant of an interaction, i. e. there is no change of state until a user action occurs or a



(a) UML metamodel of our UI profile



(b) Stereotypes to integrate multimodal UI aspects to UML state (left to right): graphical elements (≪GUI≫), speech grammar (≪Grammar≫) and system prompts (≪Prompt≫) into states.

Figure 1 Our UI profile in Fig. 1(a) and an example of its application in Fig. 1(b), which is from Dausend and Poguntke [10], but slightly differs.

running system activity is finished. In case of a state with a ≪GUI≫ stereotype, the referenced GUI element, e. g. a list, remains in the current UI representation until its state is exited, e. g. by selecting an item in the list.

We implement the stereotype ≪GUI≫ for the graphical representation of an UI element by introducing the property *gui_description* of **GUI_Component** (see Fig. 1(a)). The formal description of a GUI element is given by a specification (cf. property *specification*). Since we assume data binding, all behavior relevant properties of the specification form a list of *properties*. A

gui_configuration is used to assign a concrete value to every property of a component. A ≪GUI≫ stereotype refers to exactly one *viewState*, i. e. describing a current representation of its **GUI_Component** by referring one **GUI_ComponentConfiguration**. Furthermore, *internalBehavior* can be used to promote modeling with predefined behavior of a **GUI_Component**, e. g. the behavior of a clock. Each **GUI_Component** can have sub components (cf. *subComponents*).

We use indirections to link data to stereotypes and to allow different formalisms because this yields at least two advantages: 1) every graphical representation, grammar, or prompt needs to be defined only once and each state can create and use its own instance; 2) the complete graphics of one application can be exchanged or a different localization can be. Additionally, we do not preset formalisms for describing modality specific content so that these can be freely chosen.

For example, A *GUI_Component* does not contain the information of it representation itself, but only references to a *File*. Thus, a file can be easily exchanged in order to use different localization.

Modeling speech as a second modality requires speech grammars for speech input and prompts for speech output.

The stereotype ≪Grammar≫ (see Fig. 1(a)) enables speech recognition support. It refers to a speech grammar by property *grammar*, which can be commonly defined, e. g. as Speech Recognition Grammar Specification [50]. We decided to use Extended Backus–Naur Form (EBNF) and therefore define the property *ebnfGrammar* of **GrammarData**. Each grammar has to be named uniquely (cf. property *id*) by choosing an intuitive identifier to simplify reuse. Complex grammars can easily be constructed by nesting existing grammars using *containedGrammar*. In Fig. 1(b), the complex grammar *phone_cmds* is referenced by the property *grammar* and inherits *enter_pin*, *make_a_call*, *dial_number*, and *redial_number*.

Prompts are defined analogously to grammar specifications. We introduce the stereotype ≪Prompt≫ and its related class **PromptData** with a property *id* and a prompt phrase *text*. Figure 1(b) gives an example for a state with a ≪Prompt≫ stereotype.

We prefer a preview of a graphical representation of a ≪GUI≫ stereotype (see Fig. 1(b)) to enhance readability of state diagrams instead of showing property values.

Dynamic Aspects

As a basis to formally define our profile's semantics, we clarify its intended meaning. First, we determine the temporal aspects of each stereotypes' behavior, second, we determine the relationship between multiple stereotypes, e. g. if extending identical meta classes, and last, we describe the relations between stereotypes' semantics and the semantics of UML state diagrams.

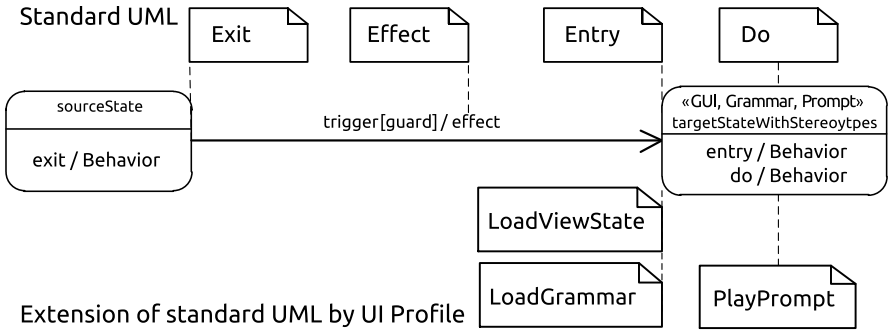


Figure 2 Process of a transition execution in temporal order (left to right) compliant to UML (upper comments) and process extensions by our UI profile (lower comments).

Figure 2 shows the different actions initiated during a state transition. The actions *Exit*, *Effect*, *Entry*, and *Do* reflect the UML semantics [39]. We define the semantics of each stereotype by introducing a corresponding static behavior (see Fig. 1(a), properties *ui_behavior*).

In the following, we refer to the behavior of each stereotype as follows: the behavior of **GUI** with *LoadViewState*, the behavior of **Grammar** with *LoadGrammar*, and the behavior of **Prompt** with *PlayPrompt*. *LoadViewState* starts after entering a state and *PlayPrompt* starts after the UML behavior *Entry* and both other stereotype behaviors have been completed. This last condition implies that the actions *LoadGrammar* and *LoadViewState* are concurrent behaviors.

A state is exited when its contained *Do* activity is completed, only if a state has an outgoing transition with neither a guard nor a trigger. This condition should be extended, so that both behaviors, *Do* and *PlayPrompt*, have to be completed before the state is exited.

4 Methodology

In this section, we explain some significant aspects of modeling concrete multimodal dialogs by applying the stereotypes of our UI profile. De Melo introduced modeling of abstract modality independent dialogs [13].

4.1 Graphical User Interface

Modeling graphic-haptic interaction using state diagrams is enhanced by our stereotype ≪GUI≫. This extension supports dynamic composition of screens at run-time. This stereotype mainly exploits two concepts of state diagrams:

- Composing state diagrams in a hierarchical manner
- Using parallelism to support active state configurations

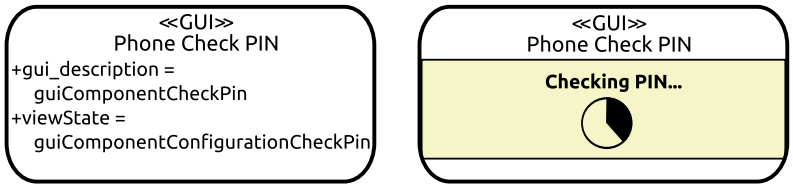


Figure 3 Alternative representations of ≪GUI≫ states.

The value of *gui_description* references a formal specification of the GUI and the value *viewState* references a widget specification of the *gui_description*.

Instead of showing a textual notation of the stereotype ≪GUI≫ inside its state, a graphical representation is used to ease readability and to assist intuitive understanding of multimodal state diagrams (see Fig. 3).

A foundational concept of state diagrams is hierarchical structuring of states by nesting. A state can be refined by adding regions with sub states. A graphical representation usually is a composition of different widgets, e. g. menus or popups, which are allowed to overlap or lay on top of each other.

Modeling dialogs of interactive systems in a hierarchical manner enables structuring dialogs in terms of tasks, or encapsulates them for re-use. This is for example useful for confirmation dialogs.

By taking into account the hierarchic level of each ≪GUI≫ state, the overall screen representation of an application is generated by composing the referenced widgets.

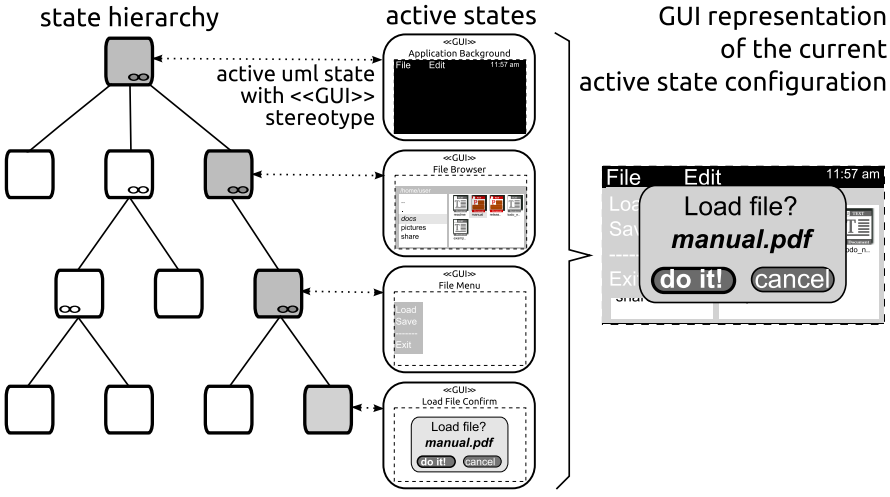


Figure 4 Composition of a screen from a stereotyped state diagram (from Dausend and Poguntke [11]).

Figure 4 illustrates how both concepts, UML state diagram hierarchy and overlay of widgets, are taken into account to compose the current screen representation of an application. The state hierarchy (see Fig. 4 - left) with its active ≪GUI≫ states (see Fig. 4 - middle) is the basis for generating the current screen representation (see Fig. 4 - right).

All active states of the state diagram are traversed starting from the topmost state of the state diagram down to the innermost active states. Starting from the topmost state, every widget of an active ≪GUI≫ state is laid on top of its parent state widget. In our example, the *Application Background* widget is overlaid by the *File Browser* widget followed by the *File Menu* widget and finally the *Load File Confirm* widget.

As a supplement to hierarchical composition, state diagrams offer the concept of orthogonal states to model concurrency. Screens often comprise multiple widgets at a time, which are independent with respect to hierarchy.

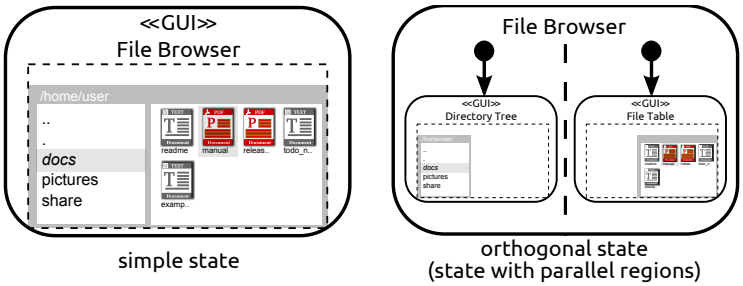


Figure 5 Compositing a screen from a state diagram with orthogonal states (from Dausend and Poguntke[11]).

Figure 5 demonstrates how the *File Browser* widget can be modeled alternatively using orthogonal states. Hence, the *File Browser* is decomposed into two meaningful components, i. e. a *File Tree* and a *File Table* view. Both parts are independent from each other and therefore can be specified own their own. Thus, the level of modularization of graphical components can freely be chosen for each graphical component as well as their behavior can specified on its own. Common behavior of both widgets is modeled at the level of their parent state.

Concurrency is a key concept for modeling multimodal interactive applications because it enables separate modeling of modality-specific aspects in a common model (cf. Sect. 7; [10]).

4.2 Speech User Interface

The speech user interface is the second important modality we take into account for modeling multimodal interactive systems. Considering input and output, the speech user interface is usually composed of speech recognition results as input and speech synthesis as output.

State-of-the-art dialog systems mainly use speaker independent speech recognition. Regarding the interaction context, the vocabulary for recognition is reduced to assure a optimal recognition rate. The speech recognition is continuously adjusted respecting the context of use represented by the overall system state. Adjusting means including or excluding vocabulary from the recognizer, e. g. some words or whole sentences.

Mapping this concept to UML state diagrams means that the context is given by an active state configuration. A simple state represents the smallest unit of a context. An active simple state with the stereotype ≪Grammar≫ extends the vocabulary of speech recognition with the referenced grammar. The concept for modeling speech recognition is closely connected to the modeling concept for GUIs.

The current vocabulary of an interactive application is composed by conjunction of all grammars of an active state configuration, i. e. all states with stereotype ≪Grammar≫. Thereby, grammars can be excluded from the vocabulary, so that the overall grammar is reduced by difference with excluded grammars (see Fig. 1(a)). The vocabulary is computed top down. In consequence, the innermost states can redefine the vocabulary (they gain higher priority than their parent states).

The stereotype ≪Prompt≫ is used to define output phrases for the speech synthesis. This stereotype extends the behavior of its state by extending its do–activity. Unlike other stereotypes, ≪Prompt≫ neither influences the active state configuration nor is influenced by it.

4.3 Multimodal Interaction

In the following, we demonstrate how multimodal interactive applications, including graphic-haptic as well as speech control, can be modeled using our prior introduced stereotypes.

Stereotypes according to UML can be freely combined with each other under two conditions: 1) The semantics of a stereotypes must not be in conflict with UML semantics; 2) The semantics of an applied stereotype must not be in conflict with semantics of other stereotypes applied to the same UML element.

For example, a state can be extended by both stereotypes ≪GUI≫ and ≪Grammar≫. This combination of stereotypes implements a special speech dialog concept, like it is used by some car manufactures: Whilst a speech dialog, a graphical representation showing a list of possible utterances is displayed to support the user to ease speech input.

Another dialog concept realized by combination of stereotypes is *barge-in*: It allows users to interrupt a system prompt by speech input. For implementation, both stereotypes ≪Grammar≫ and ≪Prompt≫ have to be applied to a same state. First, on entering such a state, the related grammar is included into the current speech recognition grammar (see Fig. 2). Next, the state's do activity is executed and its prompt is played. Here, the grammar is still active at this time. In consequence, an utterance of the user can cause exiting the state. In this case, both, execution of the do activity as well as playing the prompt, are interrupted in the course of leaving the state.

5 Formal Semantics of Profile Behavior

Before we describe the semantics of the profile, we explain its foundations. Next, we formally define the behavior, i. e. dynamic aspects, of the UI profile. The major idea is to use and pragmatically extend an existing operational semantics for state diagrams (cf. Sect. 5.1). In Sect. 5.2, we outline a more general solution for integrating the semantics of profiles and their behavior into UML in an abstract-oriented fashion.

Because "UML itself has no formal semantics for their metaclasses" [41], we base on Kohlmeyer [31], who clarifies many significant ambiguities of UML and provides a formally defined semantics. He incorporates formal semantics of state diagrams by Börger and Cavarra [4] as well as Dausend [9]. Moreover, Kohlmeyer's formal semantics includes major parts of the UML language units <Common Behaviors>, <Actions>, <Activities>, <Interactions>, <Communication> and <Use Case> of the UML to define a common semantics.

Like Kohlmeyer [31], we define the semantics of our UI profile by using ASMs [24]. Although other formal semantics exist, e. g. for UML state diagrams [16], we prefer ASMs mainly for two reasons: First, its notation is similar to pseudo-code and therefore easy to understand and second it can be directly executed, e. g. using CoreASM [15].

ASMs can be read as "pseudo-code over abstract data" [5] and comprise transition rules operating on a state composed of functions defined over a base set.

The update rule $f(s_1, \ldots, s_n) := t$ modifies the value of f at $(s_1, \ldots, s_n)$ to t. Further constructs include abstraction using *let* ... *in*, multi-way, conditionals, and rule calls with call-by-name semantics. Updates accumulated by rules are performed in parallel unless using *seq*.

We represent individual executions of a behavior by different agents. Their interaction and signal handling is modeled using (shared) ASM domains and functions [31]. The UML semantics is formally defined by ASM rules acting on these elements. Rules, which are executed by particular agents. Full details of this approach are described in [45] and [31].

According to [31], any model execution is started by an initial agent executing a rule called STARTNEWBEHAVIOREXECUTION in a particular context. This context, a *BehavioredClassifier*, is instantiated simultaneously and includes all necessary information for the execution of a behavior. Depending on the kind of behavior to execute, e. g. *StateMachine*, a corresponding rule is executed, i. e. STARTSTATEMACHINEEXECUTION. This rule instantiates a new agent for every region of a state diagram, because the behavior of different regions is performed concurrently within the same context.

The current `dispatchedEvent` is chosen from the `eventPool` of the context by the top-level agent (TLA) of a state machine execution. Each of its sub agents continuously looks for a `dispatchedEvent` or `completionEvent` and a corresponding transition to take.

```
1   PerformTransition ≡
2     if Continue then
3       case self.currentTask of
4         exit:
5           Exit(self.currState)
6         effect:
7           Effect(transition)
8         entry:
9           LoadViewState(self.tLA)
10          LoadGrammar(self.tLA)
11          Entry(self.currState)
12        do:
13          PlayPrompt(self.currState)
14          Do(self.currState)
15        finish:
16          Finish(transition)
17      else ...
```

Listing 1 ASM rule PerformTransition which defines the formal semantics of processing a transition between two states. This rule includes ASM rule calls implementing the semantics of the UI profile.

If an agent finds such a transition, this agent executes PerformTransition (see Lst. 1). If `Continue` evaluates to **true**, the case-statement is executed. The predicate `Continue` is used to guarantee that the next behavior in a state transition only starts if the previous behavior was been completed. The updates of `currentTask`, the currently executed state `currState`, and the predicate `Continue` are computed by executing agent *self* and result in the same update set as the concurrently executed rule PerformTransition.

The standard UML transition is reflected in corresponding ASM rules Exit, Effect, Entry, Do, and Finish (cf. Lst. 1, ll. 5, 7, 11, 14, and 16). These rules are described in detail by [31]. The Finish rule completes the computation and has no analogous UML element.

5.1 Direct extension of UML semantics

In order to implement the desired behavior of our UI profile (see Fig. 2), we introduce the rules LoadViewState, LoadGrammar, and PlayPrompt (cf. Lst. 1, ll. 9, 10, and 13). These rules implement the static behavior of each stereotype referenced by its properties *ui_behavior*. LoadViewState as well as LoadGrammar calculate the graphical representation and the active grammar of TLA's current active state configuration of the currently executed state. Therefore, the parameter for both rules is *self*.`tLA`. PlayPrompt implements a local behavior for the current state, so that its parameter is *self*.`currState`.

On entering a ≪Grammar≫ state the referenced grammar has to be loaded into the speech recognizer, that has to be activated as well. This has to happen in parallel to the entry behavior so that a coexisting *viewState* can be displayed simultaneously. We add the rule LOADGRAMMAR to the CASE-block **entry** as well as LOADVIEWSTATE so that they are performed in parallel with the rule ENTRY.

We utilize the hierarchy of state diagrams in the implementation of the ASM rules LOADVIEWSTATE and LOADGRAMMAR to formally define the semantics for the case of multiple active states containing UI profile's stereotypes.

If more than one state associated with ≪GUI≫ is active during a state diagram execution, the current graphical UI representation is composed by at least two viewStates (cf. Sect. 4.1). The hierarchy level of a state (incrementing from zero starting with the root state) indicates the z-ordering of its UI element inside the overall representation. Additionally, the temporal order of state activation is used to solve the problem of displaying content of orthogonal states on the same level of the state diagram hierarchy.

In the following, we illustrate the rule LOADVIEWSTATE (cf. Lst. 2) implementing the behavior of the ≪GUI≫ stereotype.

LoadViewState

The rule LOADVIEWSTATE (cf. Lst. 2) makes use of the rules GETWIDGET, COMPOSE, and RENDER. These rules define an abstract interface to our execution environment. GETWIDGET instantiates and returns a widget `viewState` from a given specification `gui_description`. COMPOSE creates and returns a list of widgets where a one list of widgets (first parameter) is laid on top of a second list of widgets (second parameter). The rule RENDER passes the given resulting list of widgets to the simulation component.

LOADVIEWSTATE is called with `tLA` as parameter. First, some local variables are declared and instantiated:

- screen — resulting list of composed widgets
- listOfWidgets — intermediately stores composed widgets
- children — sequence of direct sub states of an active states
- activeStates — sequence of current states during collection

The collecting of widgets from states with stereotypes ≪GUI≫ is processed as breadth-first search through the current active state configuration of the `tLA`. Therefore, ITERATE is executed for each level of the tree of active states. For every set of activeStates every state st with ≪GUI≫ stereotype is accessed within the FORALL. For these states, an instance of their widgets is stored in the listOfWidgets. Furthermore, all children of a state st are assigned to activeStates for the next iteration. Therefore, a case-by-case analysis is necessary to take into account that direct sub states of st can be owned by either the same agent as st or its sub agents.

```
1   LoadViewState(tLA) ≡
2   LOCAL screen := undefined, listOfWidgets := [],
        children := {},
3     activeStates :=
4     { s | s IN tLA.currState WITH
5       s.upState.region = tLA.region }
6   IN
7     /** breadth-first search over all active states */
8     ITERATE
9       FORALL st IN activeStates DO
10        /** get widget from specification */
11        IF GUI ∈ st.appliedStereotypes THEN
12          widget := GetWidget(st.gui_description,
                st.viewState)
13          ADD widget TO listOfWidgets
14        SEQ
15          children := {}
16        SEQ
17          /** get children of current state */
18          IF st.IsSimple THEN SKIP
19          ELSE IF |st.region| = 1 THEN
20            CHOOSE dst IN st.agentInState.currState
21              WITH dst.upState = st
22              ADD dst TO children
23          ELSE
24            FORALL agent IN st.agentInState.subAgents
                DO
25              CHOOSE dst IN agent.currState
26                WITH dst.upState = st
27                ADD dst TO children
28      ENDFORALL
29      SEQ
30        activeStates := children
31    ENDITERATE
32    SEQ
33      /** compose widgets to screen */
34      ITERATE
35        screen := Compose(listOfWidgets.head, screen)
36        REMOVE listOfWidgets.head FROM listOfWidgets
37      ENDITERATE
38    SEQ
39      /** display screen */
40      Render(screen)
```

Listing 2 ASM rule LoadViewState

After all states are processed, the listOfWidgets is used to Compose the screen associated with the current active state configuration in FiFo-order. Last, Render is used display the screen.

LoadGrammar

In LOADGRAMMAR, we compute the current grammar of an active state configuration similarly to the current graphical representation. It is constructed by union (if adding vocabulary) and difference (if excluding vocabulary) of all grammars referenced by active states using the stereotype ≪Grammar≫. If a sub state with this stereotype is entered, its grammar is added to or removed from the current grammars, respectively.

An example, comprising the computation of a current graphical representation and grammars for an active state configuration, is given in Sect. 7.2.

PlayPrompt

The semantics of ≪Prompt≫ defines when a prompt has to be played and, if necessary, interrupted. As intended, a prompt has to be played after all entry behaviors of a state are finished. Therefore, a prompt extends the *Do* activity of a state by playing the referenced prompt. Thus the ASM rule PLAYPROMPT (cf. Lst. 1, l. 13) has to start its own agent for playing the prompt.

If a default transition exists, i. e. a state has an outgoing transition with neither a trigger nor a guard, this state has to be exited as soon as both agents executing the rules PLAYPROMPT as well as DO (cf. Lst. 1, ll. 13 and 14) have been finished. Therefore, the ASM rule NEXTTASK of their parent agent have been modified to consider that both agents have to be finished before shifting `self.currentTask` to **finish** and setting `Continue` to **true**.

This pragmatic approach of defining a profiles semantics works but results in changes in the original UML semantics of [31]. It does not support an independent definition of the UML semantics on the one hand, and a profiles semantics on the other. In case of changes to the UML semantics, it is entirely unclear how the semantics of an extension should be adjusted. A direct embedding of the semantics description of a profile in the UML semantics relieves only the initial consistent construction of a common behavior.

5.2 Aspect-oriented extension of UML semantics

Instead of directly changing the UML semantics of Kohlmeyer [31], there are other options to achieve a more general solution. We consider the following two alternatives: 1) defining Points in the UML semantics (like anchors), where it can be extended by including additional behavioral semantics. 2) At any time while interpreting the UML semantics it has to be reviewed, whether a profile implements a semantics that must be executed at the current state. The second alternative can be understood in the sense of aspect-oriented UML extension. Since we conclude that the second approach offers a more general solution, we focus on this approach.

Aspect-oriented programming has been described first in the late 1980s and is still being developed. The most prominent implementation based on Java is

AspectJ [49], but there are also other implementations for different languages. Regarding UML, there is some work on model-driven, aspect-oriented software development [33, 2, 7]. However, to our knowledge there are no approaches using aspect-orientation in order to extend UML behavior by profiles.

A program flow is determined by sequences of expressions and jumps. An aspect can be seen as a template that adds additional instructions and jumps to such a specific program flow without changing the original program. Therefore, four components are mandatory in aspect-oriented programs: 1) *aspects*, 2) *join-points*, 3) *pointcuts*, and 4) *advices*.

A *join point* is a well defined point of a program execution. This can be a writing or reading access to a variable, a method call, a method execution, etc. A set of *join points* is called a *pointcut*. *Advices* define statements that are executed if a *pointcut* is reached during program execution. Statements of an *advice*, i. e. additional code, can be executed *before*, *around*, or *after* a *pointcut* is executed (where *around* means instead of the *pointcut*).

The central idea of aspect-orientation is the separation of concerns. The original program and its extension by *aspects* are separately implemented and later *woven* into a combined program. *Weaving* means, that a given program is analyzed for matching *pointcuts* for a set of *aspects*. If a *pointcut* is found, the statements of its corresponding *aspect's advice* are inserted as required by the *advice*, e. g. before the corresponding *join point*.

In our approach we use an aspect-oriented operational description of the specification of the semantics of a given behavior and its extensions. This semantics should be executable by its own without weaving a new combined ASM specification. Therefore, UML semantics [31] is modified to enable extensions to UML behavior in an aspect-oriented manner.

Since we have to detect *join points* during interpretation, e. g. finding method calls which usually are detected by the *weaver*, we make all *join points* explicit in our ASM specification. We do this by introducing an indirection rule for each kind of *join point*.

In the following, we explain our aspect-oriented approach for UML extension by means of our running example: the UI profile.

The implementation of our UI profile (cf. Sec. 3.1) requires method call *join points*, i. e. rule calls in ASM. Therefore, we replace each rule call in our ASM specification with rule calls of the form: CallJP(OriginalRule, [$\mathbf{arg}_1$, ..., $\mathbf{arg}_n$]), where the list $\mathbf{arg}_1$ to $\mathbf{arg}_n$ is the signature of the OriginalRule.

Next, we introduce four ASM domains and functions according to the components of aspect-oriented programs consisting with [25] (cf. Lst. 3): *Aspect*, *JoinPoint*, *Pointcut*, and *Advice*. We define the static structure for aspects-oriented programs in ASM as follows:

Each *aspect* of the corresponding domain will be assigned a `name` and also `join_points`, `pointcuts`, and `advices` as shared functions (cf. Lst. 3, ll. 3–5). The * operator indicates a list of elements of a given domain.

```
1  DOMAIN Aspect
2  SHARED name : Aspect → STRING
3  SHARED join_points : Aspect → JoinPoint*
4  SHARED pointcuts : Aspect → Pointcut*
5  SHARED advices : Aspect → Advice*
6
7  DOMAIN Pointcut
8  SHARED name : Pointcut → STRING
9  SHARED join_points : Pointcut → JoinPoint*
10
11 enum JoinPointType = {callJP , execJP ,...}
12
13 DOMAIN JoinPoint
14 SHARED name : JoinPoint → STRING
15 SHARED type : JoinPoint → JoinPointType
16 SHARED signature: JoinPoint → FILE*
17
18 enum AdviceType = {before , concurrent ,... , after}
19
20 DOMAIN Advice
21 SHARED name : Advice → STRING
22 SHARED pointcuts : Advice → Pointcut*
23 SHARED type : Advice → AdviceType
24 SHARED statement : Advice → RULE
```

Listing 3 Static structure of aspect-oriented operational semantics

A *pointcut* comprehends one or more *join points* as alternatives. Hence, a *pointcut* is relevant for executing an *aspect* if one of its *join points* is reached during program execution.

Join points are of a given `type`, e. g. **callJP**. Its `signature` defines a pattern to decide if a called function of a program matches a *join point*.

The domain *Advice*'s function `pointcuts` is mandatory for the execution of the *Rule* assigned to the function `statement`. This ASM rule has to be executed at a certain time of program execution depending on the *advice*'s function `type`. We consider the advice types **before**, **concurrent**, and **after**, whereas the type *around* is not taken into account, as it means executing an *advice's* `statement` instead of a rule call of the UML semantics. Thus, the semantics of UML is changed improperly, which is forbidden by using the profile mechanism (cf. Sect. 2.4).

The original rule and its signature are assigned as parameters par_rule and par_sig to the rule call CALLJP (Lst. 4, l. 5) to implement indirection of rule calls. CALLJP compares the given signature with the signatures of each join point of the domain *JoinPoint*. If both signatures match, the rule EXECUTION is called to execute the corresponding *advices* of the matching *aspects*.

A *join point* defines a pattern as a list $[\mathbf{arg}_1, \ldots, \mathbf{arg}_n]$. Each entry of the list is either a concrete *value* or **undefined**, where **undefined** stands for *any*

value. Therefore, two corresponding entries of the signature and the pattern match if they are equal or if the entry of the pattern is **undefined**.

```
1  RULE CallJP(par_rule, par_sig) ≡
2    SEQ
3      jpts:={}
4      FORALL jpt ∈ JoinPoint DO
5        IF match(par_sig, jpt.signature) THEN
6          FORALL asp ∈ Aspect WITH
7            ∃ jpt ∈ join_point(asp) DO
8              ADD jpt TO jpts
9          ENDFORALL
10     ENDFORALL
11   NEXT
12     Execution(findAdvices(findPointcuts(jpts)),
           par_rule, par_sig)
```

Listing 4 The ASM rule CallJP is called for each rule of a ASM program

Lst. 5 shows the derived function *match*. It constructs a set of matching pairs of signatures and patterns by set comprehension, and returns **true** if this set is not empty and **false** otherwise.

```
1  DERIVED FUNCTION match(par_signature, par_pattern) =
2    IF {matches IS {(s,p)} |
3      i ∈ [1..par_signature.length],
4      s ∈ par_signature, p ∈ par_pattern :
5        par_signature.length = par_pattern.length ∧
6        par_signature[i] = s ∧
7        par_pattern[i] = p ∧
8        (s = p ∨ p = undefined)} ≠ {}
9    THEN result := true
10   ELSE result := false
```

Listing 5 Predicate rule indicating a match of a rule's signature with a join point's pattern.

We collect matching *join points* of any *aspect* (cf. Lst. 4) in the set jpts. Relevant *pointcuts* and *advices* are computed by set comprehensions (cf. Lst. 6).

```
1  DERIVED FUNCTION findPointcuts(jpts) =
2    { ptcs IS ptc | ptc ∈ Pointcut, jpt ∈ jpts
3      WITH jpt ∈ ptc.join_point }
4
5  DERIVED FUNCTION findAdvices(ptcs) =
6      { advs IS adv |  adv ∈ Advice
7        WITH ∀ ptc ∈ adv.pointcut : ptc ∈ ptcs }
```

Listing 6 Derived functions using set comprehension to find relevant pointcuts/advices for a given set of join points/pointcuts.

The rule EXECUTION (cf. Lst. 7) calls each statement of the given advices **p1_advs** at the right time, based on each *advice's* type. Since,*advices* can only have local effects, i. e. within the current update set [5], we insert additional ASM rules by using sequential blocks to implement aspects with the types **before**, **concurrent** and **after**. Using sequential blocks enables extending an ASM specification in a way that the ordering of original and introduced rules is taken into account and however all effects result in the same update set of the current step of the executing agent.

```
1 RULE EXECUTION(par_advs, par_rule, par_sig) ≡
2   SEQ
3     self.mode := before
4   NEXT
5     ITERATE
6       /** execute advice's statement */
7       FORALL adv IN par_advs WITH
8           adv.type = self.mode DO
9         adv.statement
10
11      /** execute original rule */
12      IF self.mode = concurrent THEN
13        par_rule(par_sig)
14
15      /** set next mode for agent self */
16      NEXTADVICEMODE(self)
17    ENDITERATE
18
19 RULE NEXTADVICEMODE(par_agent) ≡
20   CASE par_agent.mode OF
21     before : par_agent.mode := concurrent
22     concurrent : par_agent.mode := after
23     after : SKIP
24   ENDCASE
```

Listing 7 Executing advices' statements and its corresponding original rule.

Initially, the mode of the executing ASM agent is set to **before** (cf. Lst. 7, l. 3). Next, inside the ITERATE-block, three actions are considered in parallel:

- The statement of any advice in par_advs is called if its advice type equals the current mode of the executing agent (cf. Lst. 7, ll. 7–9).
- The rule par_rule is executed if the mode of the executing agent is **concurrent** (cf. Lst. 7, ll. 12–13).
- the rule NEXTADVICEMODE (cf. Lst. 7, l. 19–24) with the executing agent as parameter is called, which sets the mode of this agent to the next mode (cf. Lst. 7, l. 16).

In the case of executing NextAdviceMode with `self.mode` equals **after**, skip is executed, i. e. an empty update rule. Since all new updates are empty, the iterate block is be finished.

The rule Execution (see Lst. 7) modifies the update set caused by the original rule depending on an advice's type and its statement as follows: The modified update set is computed by applying the definition in Tab. 1 to the rule Execution. If there exists any advice with type **before**, the update U is a result of the advices' statements which are a part of P. In the next iteration step (cf. Lst. 7) Q yields V where Q is a composition of statements for all advices of type **concurrent** and the original rule.

Table 1 Definition of the semantics of ASM sequential rules from Börger and Stärk [5, p. 74]

$$\frac{yields(P,\mathfrak{A},\varsigma,U) \quad yields(Q,\mathfrak{A}+U,\varsigma,V)}{yields(P \textbf{ seq } Q,\mathfrak{A},\varsigma,U \oplus V)} \quad \text{if } U \text{ is consistent.}$$

where

$$U \oplus V = V \cup \{(l,v) \in U \mid \text{there is no } w \text{ with } (l,w) \in V\}$$

is a composition of update sets.

$$\frac{yields(P,\mathfrak{A},\varsigma,U)}{yields(P \textbf{ seq } Q,\mathfrak{A},\varsigma,U)} \quad \text{if } U \text{ is inconsistent.}$$

The definition can be applied on the advices of type **after** just as it was shown in the previous case.

Since the definition (cf. Tab. 1) requires both sets to be consistent, we can guarantee that the updates caused by the original rule are not contradicted by any of the advices' statements.

Although we introducing indirections CallJP we guarantee that each of the original ASM rules is performed at the right time (cf. Lsts.4, 7) and we do not rewrite any original rule. So we guarantee that the semantics of the extension does not contradict the UML semantics in the current update step under the following confinement: However, the UML semantics can be harmful interfered by changing some of its shared functions. Thus changing one of those shared functions, any extension of UML has to guarantee by itself that UML semantics is just extended and not changed leading to contradictions between semantics of UML and its extension.

Listing 8 shows the definition of our UI extension based on the aspect-oriented approach in contrast to our direct approach in Sec. 5.1. Our UI extension is modeled as one *Aspect* containing *advices* and *pointcuts* with *join points* to extend the UML and its semantics by our aspect-oriented extension.

The rules LoadGUI, LoadGrammar, and LoadPrompt are omitted. They implement the semantics as explained in Sec. 5.1. For example, the rule LoadGUI is similar to the rule LoadViewState (cf. Lst. 2).

```
1  INITIALIZE ≡
2    SEQ
3      JoinPoint(jp-entry):= true
4      name(jp-entry) := ENTRY
5      type(jp-entry) := callJP
6      args(jp-entry) := undefined
7
8      JoinPoint(jp-do) := true
9      name(jp-do) := DO
10     type(jp-do) := callJP
11     args(jp-do) := undefined
12
13     Pointcut(pc-gui) := true
14     name(pc-gui) := PoCuGUI
15     ADD jp-entry TO join-points(pc-gui)
16
17     Pointcut(pc-grammar):= true
18     name(pc-grammar) := PoCuGrammar
19     ADD jp-entry TO join-points(pc-grammar)
20
21     Pointcut(pc-prompt) := true
22     name(pc-prompt) := PoCuPrompt
23     ADD jp-do TO join-points(pc-prompt)
24
25     Advice(ad-gui) := true
26     name(ad-gui) := Ad–GUI
27     pointcuts(ad-gui) := pc-gui
28     type(ad-gui) := before
29     statement(ad-gui) := LoadGUI
30
31     Advice(ad-grammar) := true
32     name(ad-grammar) := Ad–Grammar
33     pointcuts(ad-grammar) := pc-grammar
34     type(ad-grammar) := before
35     statement(ad-grammar) := LOADGRAMMAR
36
37   Advice(ad-prompt) := true
38     name(ad-prompt) := Ad–Prompt
39     pointcuts(ad-prompt) := pc-prompt
40     type(ad-prompt) := concurrent
41     statement(ad-prompt) := PLAYPROMPT
42   NEXT
43     Aspect(asp-ui) := true
44     name(asp-ui) := Asp–GUI
45     ADD ad-gui TO advices(asp-ui)
46     ADD pc-gui TO pointcuts(asp-ui)
47     ADD ad-grammar TO advices(asp-ui)
48     ADD pc-grammar TO pointcuts(asp-ui)
49     ADD ad-prompt TO advices(asp-ui)
50     ADD pc-gprompt TO pointcuts(asp-ui)
```

Listing 8 ASM rule to initialize the aspect-oriented UI extension.

6 Application: Multimodal PIN Entry

In this section, we give an example of applying our modeling approach for multimodal interactive systems. First, we describe a UC. Then we will explore how the dialogue provided by this UC can be modeled with our approach.

6.1 Introducing the Use Case

Our UC takes place in an in-car scenario. The vehicle provides a numeric keypad for manual interaction as part of its onboard infotainment system as well as some buttons on the steering wheel. One of the keys called Push-to-Activate (PTA) can be used to activate a speech dialog. This enables multimodal interaction.

The scenario can be summed up from the viewpoint of the system as follows: 1. A phone is being inserted, 2. the system waits for PIN entry, 3. the entered PIN is checked, 4. the phone is activated (if the PIN is correct). From the user point of view, the PIN can be entered via the numeric keypad or a speech dialog. In the following, we assume a special UC: Entering the first two digits of the PIN by hand and continuing pin entry by speech dialog, because of a significant change of the driving situation.

6.2 Modeling Multimodal PIN Entry

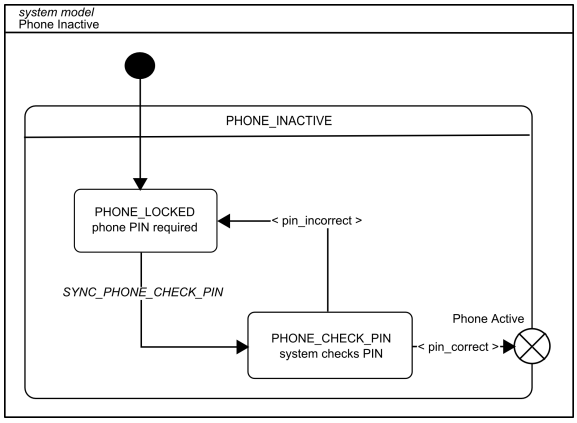
The pin entry scenario starts after the mobile phone is plugged into a cradle, but the phone is still locked. The system model (see Fig. 6(a)) contains all states of the system, transitions and events, which are relevant for the HCI of the UC described in section 6.1. The transition from the state PHONE_LOCKED into PHONE_CHECK_PIN is executed if the event SYNC_PHONE_CHECK_PIN is dispatched. The diagram will be left by an exit point, if the pin is correct.

The multimodal model is structured into parallel regions, one for each modality and one for the system model. A screen sequence (see Fig. 6 (b)) and the diagrams according to our pin entry UC are shown in Fig. 6 ((a), (c) & (d)).

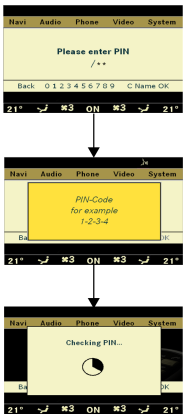
The graphic-haptic part of the dialog comprises the states ENTER_PIN and CHECKING_PIN, both extended by a ≪GUI≫ stereotype. Their related UI element is displayed while they are active. The sub state of ENTER_PIN describes the input of numbers via the keypad of the car.

The multimodal dialog starts with activating the state ENTER_PIN, including its ≪GUI≫ stereotype and its sub states. Now, the user enters the first two digits by hand using the numeric keypad.

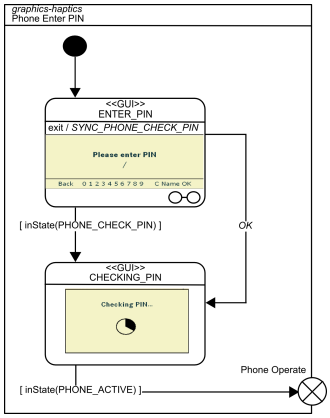
Next, the speech dialog is activated using the PTA-button on the steering wheel and the state SPEAK_PIN is entered (see Fig. 6 (d)). This state contains all kinds of stereotypes from our UI profile. According to our formal semantics



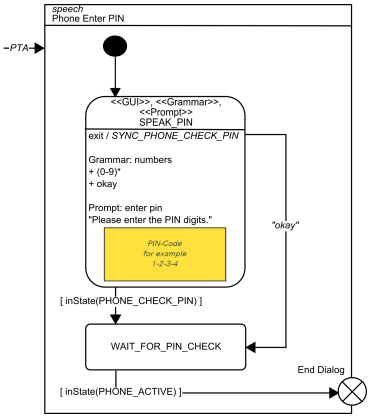
(a) relevant part of the system model



(b) display



(c) haptic interaction



(d) speech interaction

Figure 6 (a): part of the common system model; (b): screen sequence of an interaction sequence for pin entry; (c): model of the haptics dialog for pin entry; (d): model of the speech dialog for pin entry. (from Dausend and Poguntke [10])

(cf. Sect. 5), the viewState — a so called Teleprompter[1] — is shown on a z-level above the viewState of ENTER_PIN. Furthermore, the grammar is loaded and the speech recognizer is started. When, loading the screen and grammar, are completed, the prompt "Please enter the PIN digits." is played.

The user says the missing numbers and confirms his input by saying "okay", which forces the state SPEAK_PIN to be exited. Its exit action is executed, firing the event SYNC_PHONE_CHECK_PIN and deactivating the Teleprompter. Afterwards the state WAIT_FOR_PIN_CHECK is entered.

The system model (see Fig. 6(a)) receives the synchronization event SYNC_PHONE_CHECK_PIN, which triggers the transition into the state PHONE_CHECK_PIN.

[1] a screen used by Mercedes Benz to visualize help, e. g. possible speech commands.

By entering this state, the guard [inState(PHONE_CHECK_PIN)] (see Fig. 6(c)) becomes *true*. This guard ensures that the system model is in the state PHONE_CHECK_PIN and triggers the necessary transition into CHECKING_PIN. The "inState" construct is defined in the Object Constraint Language [40].

After checking the PIN, the system model changes its state, if the PIN is correct, into the state PHONE_ACTIVE whereupon both models are synchronized executing their transition towards their exit states.

7 Validation of the Modeling Concept

The validation of our modeling approach consists of two parts. The first part is conducted by case studies during the development process (cf. Sect. 3) and the second part is an expert evaluation which has been performed after development had reached a solid basis.

First, we have implemented models for selected UCs for different interaction aspects of multimodality as a proof of concept. Thereby, we compared two different kinds of modeling HCI with a focus on the synchronization between modalities. The implementation has been supported by our own modeling tool and interpreter based on our formal semantics.

The second part of our validation is an expert evaluation. This evaluation answers the question in terms of an qualitative analysis:"How can a user-friendly method for the specification of multimodal interaction be achieved?". We will describe the scenario used in our expert evaluation as an example of an application of our approach which is based on the aforementioned use case (cf. Sect. 6.1).

7.1 Case study

The case study consists of several proofs of concept. We use real parts of dialogs of the recent Mercedes Benz C-Class for validation. We have modeled a large part of the telephone application for graphic-haptic operation and speech operation, checked its UML conformity and simulated its behavior as well as its UI. This simulation is facilitated by our own interpreter, which supports modeling, UML conformance checking, and executing models including its resulting UIs by simulation of UML state diagrams and our UI profile.

As mentioned above, [23] provides a proof of concept for modeling multimodal dialogs using state machines with events for synchronization between different modalities. This method limits extensibility, because the number of events needed for a single synchronization corresponds to the number of modalities, which need to be synchronized. Moreover, it has to be considered in which state and under what condition a modality should be responsible for a particular synchronization event.

In order to improve this situation, we have analyzed two principal approaches to model multimodality: First, modeling one abstract dialog and integrating speech and graphic-haptic dialogs into it. Second, separate modeling of each modality based on a common system model for synchronization.

We favor the second method: The modeling method follows the model view controller (MVC) architecture, so that system model (*model*), modalities (*controller)*, and its representation (*view*) are decoupled from each other.

This modeling strategy leads to some redundancy, because modalities follow the same abstract dialog. This strategy otherwise facilitates flexibility regarding extensibility of functions as well as modalities.

In the following, we present some data of our most comprehensive case study, using a common system model for synchronization: 158 viewStates were defined in total, including 29 prompts as well as grammars for speech operation of the telephone associated functions. Three kinds of state diagrams were modeled: 16 for graphic-haptic operation, 4 for modeling of UI relevant system parts and 19 for speech operation. The model contains 206 states in total (without pseudo states) and 546 transitions.

As a result of this case study we have got extensive collection of state diagrams and their UML model including our stereotypes from the UI profile. The behavior specified by our model as well as simulation of the UI is similar to the Mercedes C-Class. We so far see no problems where modeling interaction scenarios of current in-car dialogs may fail in principle with our approach. However, we think that the modularization of models is very important for different aspects like readability, understandability, and reusability.

7.2 Expert Evaluation

In this section, we present results of our expert evaluation [10]. In this evaluation, we have answered the following research question: "How can a user friendly methodology for specification of multimodal interaction be achieved?" We have formulated further detailed questions regarding appropriateness and usability:

Appropriateness

- Are UML state diagrams extended with stereotypes appropriate to model interactive applications?
- Is the synchronization of modalities by using a common system model appropriate to model multimodality?

Usability

- Is our approach effective, efficient, easy to learn and less error-prone?
- Are the resulting models of our approach easy to read and understand?

Methodology

Based on our research question, we have refined questions related to appropriateness of the UML approach, our extensions using profiles and stereotypes, and synchronization of modalities using a common system model. Furthermore we want to gain knowledge about ergonomic aspects like effectiveness, efficiency, learn-ability, and error rate. The questions are answered by an expert evaluation.

We have chosen the following participants to ensure that all experts have knowledge of UML as well as HCI: Scientific staff of the Institute of Software Engineering and Compiler Construction and the Institute of Media Informatics, both of Ulm University, and employees of Daimler AG R&D in Ulm.

We decided for an oral interview without predetermined answers to receive detailed comments, suggestions and personal statements of each participant. The resulting non numeric material, the interview protocols, provide more details than a quantitative reading/measurement would contain. Furthermore for analysis we profit from content-diversity of individual answers [3]. Based on [12] and [14], we expected high effectiveness of this elicitation method for our purpose.

Experiment Design

The evaluation starts with collecting personal data: age, gender, profession or job, and knowledge of UML and HCI. Second, we introduced all participants to our interaction scenario. Last, we interviewed every participant and transcribed their thoughts, questions, comments, and remarks said aloud.

After collecting the personal data, the evaluation starts with a short introduction of each participant to our scenario, its environment, and interaction opportunities. We choose an in-car multimodal phone dialog where the goal of this scenario is to unlock a mobile phone via the car's user interface (see Sect. 6.2).

The scenario was explained by means of a presentation whereas modeling the scenario was explained by using printouts. Next, the participants were given 10 minutes time to model an extension of the scenario on paper using the newly learned concepts. They were asked to express their thoughts loudly.

The participants should supplement the model by an incoming call, accepting the incoming call, as well as retrieving caller information. Therefore, we gave each participant a presentation of the extended UC, the corresponding system model, fixed parts of the two models for voice control and graphic-haptic operation, as well as the necessary stereotypes. Finally, an interview of 10 to 15 minutes was conducted.

Expert Evaluation Results

The summary of our main results is categorized as follows: UML and multimodality and general user-friendliness.

The participants of our study have been in the age of 23 to 33 years. Of the 11 participants, eight are male and three are female. Four are graduates of computer science, five are graduates of engineering, and two are advanced computer science students. The number of participants is sufficient, because analyzing expert opinions does not require any quantitative analysis.

To determine the experience with UML and HCI every participant is interviewed and classified by us on a Likert-scale from one to seven, where one means "no experience" and seven means "very much experience". All except two participants self-estimate their knowledge of UML as well as HCI at least as respective, most of them assert to have above average experience.

For the extraction of results from our protocols, the qualitative content analysis was chosen because it is especially suitable for the evaluation of the available interviews [21]. This systematic procedure means searching on the protocols with an analytical grid for relevant information. The resulting information is processed as far as possible independently from the protocols to answer the research question.

The analysis grid in the present investigation includes the topics and aspects from Sect. 7.2. Also, all protocols have been analyzed with regard to proposals for improvements of our approach.

UML and Multimodality

All participants accepted UML for modeling *multimodality* without hesitation. Six participants state that the modality independent system model is appropriate. Two participants highlight clearness and understandability, four comment positively, but have minor remarks for improvements. The introduction to the model is quite difficult and a more sophisticated model could be hard to model. As an example, participant H cites the destination entry for navigation. A problem might also arise if system conditions that are used only for specific modalities would be needed. A majority of nine participants believe that the system model should be present as a basis for modeling multimodality. A parallel creation is possible for professionals. Alternatively, a modality could first be specified from which the system model can be partly derived. And participant F remarks that he had initially not fully understood the system model.

Six participants state that the stereotypes are simple to understand, easy to use, good to illustrate dialogs and that they can be used in a modular way. Four participants had problems with the grammar stereotypes. Using the grammar requires profound knowledge of its semantics. Participant F thought, that the grammar stereotype has to be applied to the destination state of a

speech dialog transition. Three participants have not taken into account the semantics of the stereotypes or the semantics was overall not clear for them.

Seven participants estimate that the concept is extensible for further modalities. Participant B states that he has no idea how to model multimodality more easily. Participant H perceives that he sees no solution for modeling combined modality input like: "Go there!" with simultaneous pointing gesture. Participant G anticipates that it could be problematic if modality-specific dialogs have different numbers of steps.

Ease of Use

The statements relating to the general user experience were classified into the categories of effectiveness and efficiency, possibility of errors and learning. Six participants confirm with the studied modeling approach multimodality can be effectively specified. Participant K admits that it is possible, if one has previously dealt with UML and the extended syntax. Participant A affirms efficiency of the approach and its intuitive usability. Participant D, however, criticizes the partial redundancy between modalities. Participant C can hardly imagine multimodality, in case of modeling both modalities at the same time.

Recommendations for Improvements

A total of seven participants want tool support to ensure consistency between the system model and the modalities and moreover provide automation, for example by offering auto-completion. Three Participants propose support by simulation of the models as well as the user interface to simulate multimodality, to get help in determining the active grammars and to compare the model effectively with present requirements. Two Participants express that the parts referring to the system model also should be highlighted in the modality-related diagrams in a different color in order to increase readability.

8 Discussion and Conclusion

Starting from the challenges for modeling multimodal interactive systems, we address the following issue: "How can a user-friendly method for the specification of multimodal interaction be achieved?". The hypotheses from Sect. 2.2 are the basis for the discussion of the presented approach. Additionally, we consider the results of the validation (cf. Sect. 7) to add some remarks.

UML is an *established formalism* in SE which is also used in the HCI community. In contrast to other approaches (cf. Sect. 2), which aim for preferably extensive support for design of interactive system, we concentrate on defining a formal semantics for our UML profile to support MDD and simultaneously take into account conformity with UML. The independence of the modalities

and the synchronization using a common system model is a great advantage, especially if dialogs of different modalities consist of different numbers of steps.

The *formal semantics* of our extension (cf. Sect. 5) is based on an existing semantics for UML behavior using ASMs. Our approach can easily be extended for other diagrams, e. g. activity diagrams, because they are already part of the semantics we build on. By providing formal semantics, we fulfill a major claim for defining UML profiles for modeling behavioral aspects (cf. Sect. 2.4).

On this basis, we try to assure *high quality of models*. Therefore, we provide *tool support* to define and apply profiles, e. g. the UI profile, to state diagrams. Other current tools allow the definition and usage of UML profiles, but only on the level of notation, i. e. diagrams. We further introduced in our tool some additional constraints to the UML constraints for state diagram models to guarantee computability of our models. The behavior of models of multimodal interactive systems as well as the representation of the UI related to an active state configuration (cf. Sect. 7) can be simulated. This makes the approach applicable to early phases of system development, e. g. to compare different multimodal interactions.

Code generation based on the formal semantics is possible, but is only used for simple models without concurrency and history states.

Introducing our UI profile and defining its formal semantics in an direct way is a good example for extending UML (cf. Sect. 3). Being inspired by real needs, it makes pros and cons of UML profiles in general evident.

Our aspect-oriented approach to extend UML semantics can be conferred to other extending ASM specifications. Currently, we analyze different kinds of ASM specifications to find out which requirements such an ASM specification should satisfy to properly fit to our aspect-oriented extension approach and how our approach can be enhanced.

The validation (cf. Sect. 7) confirms that our approach meets both goals: modeling multimodality while keeping general user-friendliness. The tool, which was not part of the expert evaluation, can help to overcome most of the problems mentioned, e. g. difficulties in understanding the profile's semantics in specific situations.

Literature

[1] H. Balzert. Wie erstellt man ein objektorientiertes Analysemodell? *Informatik-Spektrum*, 20(1):38–47, 1997.

[2] M. Basch and A. Sanchez. Incorporating Aspects into the UML. In *Proceedings of Third International Workshop on Aspect-Oriented Modeling*, 2003.

[3] J. Bortz and N. Döring. *Forschungsmethoden und Evaluation für Human- und Sozialwissenschaftler*. Springer, 2006.

[4] E. Börger, A. Cavarra, and E. Riccobene. Modeling the Dynamics of UML State Machines. *Abstract State Machines, Theory and Applications, International Workshop, ASM*, 1912:19–24, 2000.

[5] E. Börger and R. Stärk. *Abstract State Machines – A Method for High-Level System Design and Analysis*. Springer, 2003.

[6] H. Bubb. *Future Applications of DHM in Ergonomic Design*, pages 779–793. Springer-Verlag, 2007.

[7] T. Cottenier, A. van den Berg, and T. Elrad. Stateful Aspects: The Case for Aspect-Oriented Modeling. In *Proceedings of the 10th International Workshop on Aspect-Oriented Modeling*, 2007.

[8] P. P. da Silva and N. W. Paton. UMLi: The Unified Modeling Language for Interactive Applications. In A. Evans, S. Kent, and B. Selic, editors, *UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference*, volume 1939, pages 117–132, York, UK, 2000. Springer.

[9] M. Dausend. Entwicklung einer ASM-Spezifikation der Semantik der Zustandsautomaten der UML 2.0. Diploma thesis, Universität Ulm, 2007.

[10] M. Dausend and M. Poguntke. Spezifikation multimodaler interaktiver Anwendungen mit UML. In *Mensch & Computer*, pages 215–224. Oldenbourg Verlag, 2010.

[11] M. Dausend and M. Poguntke. Ausführbare UML-Modelle multimodaler Interaktionsanwendungen. *i-com: Zeitschrift für interaktive und kooperative Medien*, 10(3):33–39, 2011.

[12] A. Davis, O. Dieste, A. Hickey, N. Juristo, and A. Moreno. Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review. In *14th IEEE International Requirements Engineering Conference*. IEEE Computer Society, 2006.

[13] G. de Melo. *Modellbasierte Entwicklung von Interaktionsanwendungen*. PhD thesis, Universität Ulm, 2010.

[14] O. Dieste, N. Juristo, and F. Shull. Understanding the Customer: What Do We Know about Requirements Elicitation? *Software, IEEE*, 25(2):11–13, 2008.

[15] R. Farahbod, V. Gervasi, and U. Glässer. CoreASM: An Extensible ASM Execution Engine. *Fundamenta Informaticae*, 77(1):71–103, 2007.

[16] H. Fecher and J. Schönborn. UML 2.0 State Machines: Complete Formal Semantics via Core State Machines. In L. Brim, B. R. Haverkort, M. Leucker, and J. van de Pol, editors, *FMICS/PDMC*, volume 4346 of *Lecture Notes in Computer Science*, pages 244–260. Springer, 2006.

[17] L. Fuentes-Fernández and A. Vallecillo-Moreno. An Introduction to UML Profiles. *The European Journal for the Informatics Professional*, 5(2):6–13, 2004.

[18] D. Gessenharter. Mapping the UML2 Semantics of Associations to a Java Code Generation Model. In K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, editors, *Model Driven Engineering Languages and Systems*, volume 5301 of *Lecture Notes in Computer Science*, pages 813–827. Springer, 2008.

[19] D. Gessenharter. Implementing UML Associations in Java – A Slim Code Pattern for a Complex Modeling Concept. In *Proceedings of the Workshop on Relationships and Associations in Object-Oriented Languages*, RAOOL '09, pages 17–24, New York, NY, USA, 2009. ACM.

[20] D. Gessenharter and M. Rauscher. Code Generation for UML 2 Activity Diagrams - Towards a Comprehensive Model-Driven Development Approach. In R. B. France, J. M. Küster, B. Bordbar, and R. F. Paige, editors, *European Conference on Modelling Foundations and Applications*, volume 6698 of *Lecture Notes in Computer Science*, pages 205–220. Springer, 2011.

[21] J. Gläser and G. Laudel. *Experteninterviews und qualitative Inhaltsanalyse als Instrumente rekonstruierender Untersuchungen.* Wiesbaden. VS-Verlag für Sozialwissenschaften, Wiesbaden, 2009.

[22] S. Goronzy and N. Beringer. Integrated Development and on-the-Fly Simulation of Multimodal Dialogs. In *Ninth European Conference on Speech Communication and Technology.* ISCA, 2005.

[23] S. Goronzy, R. Mochales, and N. Beringer. Developing Speech Dialogs for Multimodal HMIs Using Finite State Machines. In *Ninth International Conference on Spoken Language Processing*, pages 1774–1777, Pittsburgh, Pennsylvania, 2006. ISCA.

[24] Y. Gurevich. *Evolving Algebras 1993: Lipari Guide*, pages 9–36. Oxford University Press, Inc., New York, NY, USA, 1995.

[25] Y. Han, G. Kniesel, and A. Cremers. A Meta Model for AspectJ. Technical report, Technical Report IAI-TR-2004-3, Computer Science Department III, University of Bonn, 2004.

[26] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[27] R. Hennicker and N. Koch. A UML-based Methodology for Hypermedia Design. *Lecture Notes in Computer Science*, 1939:410–424, 2001.

[28] R. Hennicker and N. Koch. Modeling the User Interface of Web Applications with UML. In *Practical UML-Based Rigorous Development Methods-Countering or Integrating the eXtremists, Workshop of the pUML-Group at the UML*, volume 200, 2001.

[29] A. Kölzer. Diamod - a tool for modeling dialogue applications. In *Proceedings of the COLING-2000 Workshop on Using Toolsets and Architectures To Build NLP Systems*, pages 69–78, Centre Universitaire, Luxembourg, 2000. International Committee on Computational Linguistics.

[30] A. Kölzer. *Diamod: Ein Werkzeugsystem zur Modellierung natürlich-sprachlicher Dialoge.* PhD thesis, Universität Koblenz, 2002.

[31] J. Kohlmeyer. *Eine formale Semantik für die Verknüpfung von Verhaltensbeschreibungen in der UML 2.* PhD thesis, Universität Ulm, 2009.

[32] J. Kohlmeyer and W. Guttmann. Unifying the Semantics of UML 2 State, Activity and Interaction Diagrams. *Perspectives of Systems Informatics*, 5947:206–217, 2010.

[33] R. P. Laurence, L. Duchien, G. Florin, F. Legond-aubry, L. Seinturier, and L. Martelli. A UML Notation for Aspect-Oriented Software Design. In *Workshop on Aspect-Oriented Modeling with UML*, 2002.

[34] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, M. Florins, and D. Trevisan. Usixml: A user interface description language for context-sensitive user interfaces. In K. Luyten, M. Abrams, Q. Limbourg, and J. Vanderdonckt, editors, *Proceedings of the ACM AVI 2004 Workshop 'Developing User Interfaces with XML: Advances on User Interface Description Languages'*, pages 55–62, Gallipoli, 2004. ACM.

[35] C. Martins and A. Silva. Modeling User Interfaces with the XIS UML Profile. In *Proc. Ninth Int. Conf. Enterprise Information Systems*, number 9, Funchal, Portugal, 2007. ICEIS, Springer.

[36] L. Nóbrega, N. Nunes, and H. Coelho. Mapping ConcurTaskTrees into UML 2.0. *Lecture Notes in Computer Science*, 3941:237–248, 2006.

[37] N. Nunes and J. Cunha. Towards a UML Profile for Interaction Design: The Wisdom Approach. *Lecture Notes in Computer Science*, 1939:101–116, 2000.

[38] Object Management Group. OMG Unified Modeling Language (OMG UML) Infrastructure v2.4.1, 2011.

[39] Object Management Group. OMG Unified Modeling Language (OMG UML) Superstructure v2.4.1, 2011.

[40] Object Management Group. OMG Object Constraint Language (OCL) v2.3.1, 2012.

[41] J. Pardillo. A Systematic Review on the Definition of UML Profiles. In D. C. Petriu, N. Rouquette, and Ø. Haugen, editors, *MODELS*, volume 6394 of *Lecture Notes in Computer Science*, pages 407–422. Springer, 2010.

[42] H. Partsch, M. Dausend, D. Gessenharter, J. Kohlmeyer, and A. Raschke. From Formal Semantics to Executable Models: A Pragmatic Approach to Model-Driven Development. *Int. Journal Software and Informatics*, 5(1-2):291–312, 2011.

[43] F. Paternò, C. Mancini, and S. Meniconi. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction*, pages 362–369, 1997.

[44] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[45] S. Sarstedt. *Semantic Foundation and Tool Support for Model-Driven Development with UML 2 Activity Diagrams*. PhD thesis, Universität Ulm, 2005.

[46] S. Sauer and S. Engels. Extending UML for Modeling of Multimedia Applications. In *Proceedings 1999 IEEE Symposium on Visual Languages*, pages 80–87. IEEE, 1999.

[47] A. Silva. The XIS Approach and Principles. In *Proceedings of the 29th Conference on EUROMICRO*, page 33. IEEE Computer Society, 2003.

[48] I. Sommerville. *Software Engineering*. Addison-Wesley, Harlow, England, ninth edition, 2010.

[49] The AspectJ Team. The AspectJ Project. www.eclipse.org/aspectj/, 2011. (last access December 2011).

[50] W3C. Speech Recognition Grammar Specification 1.0. www.w3.org/TR/speech-grammar/, 2004. (last access Dezember 2011).

[51] W. Wahlster. *SmartKom: Foundations of Multimodal Dialogue Systems*. Springer, 2006.

Liste der bisher erschienenen Ulmer Informatik-Berichte

Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm

Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

91-02* *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

91-03* *Alfons Geser*
Relative Termination

91-04* *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

91-06* *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

91-07* *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

91-08* *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

92-01* *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

92-02* *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluation of Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

92-04* *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

92-05* *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

92-06* *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

92-07* *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

92-08* *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

92-09* *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

92-14* *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers

94-05 *V. Arvind, J. Köbler, R. Schuler*
On Helping and Interactive Proof Systems

94-06 *Christian Kalus, Peter Dadam*
Incorporating record subtyping into a relational data model

94-07 *Markus Tresch, Marc H. Scholl*
A Classification of Multi-Database Languages

94-08 *Friedrich von Henke, Harald Rueß*
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

94-09 *F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker*
Construction and Deduction Methods for the Formal Development of Software

94-10 *Axel Dold*
Formalisierung schematischer Algorithmen

94-11 *Johannes Köbler, Osamu Watanabe*
New Collapse Consequences of NP Having Small Circuits

94-12 *Rainer Schuler*
On Average Polynomial Time

94-13 *Rainer Schuler, Osamu Watanabe*
Towards Average-Case Complexity Analysis of NP Optimization Problems

94-14 *Wolfram Schulte, Ton Vullinghs*
Linking Reactive Software to the X-Window System

94-15 *Alfred Lupper*
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

94-16 *Robert Regn*
Verteilte Unix-Betriebssysteme

94-17 *Helmuth Partsch*
Again on Recognition and Parsing of Context-Free Grammars:
Two Exercises in Transformational Programming

94-18 *Helmuth Partsch*
Transformational Development of Data-Parallel Algorithms: an Example

95-01 *Oleg Verbitsky*
On the Largest Common Subgraph Problem

95-02 *Uwe Schöning*
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

95-03 *Harry Buhrman,Thomas Thierauf*
The Complexity of Generating and Checking Proofs of Membership

95-04 *Rainer Schuler, Tomoyuki Yamakami*
Structural Average Case Complexity

95-05 *Klaus Achatz, Wolfram Schulte*
Architecture Indepentent Massive Parallelization of Divide-And-Conquer Algorithms

95-06 *Christoph Karg, Rainer Schuler*
Structure in Average Case Complexity

95-07 *P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe*
ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen

95-08 *Jürgen Kehrer, Peter Schulthess*
Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik

95-09 *Hans-Jörg Burtschick, Wolfgang Lindner*
On Sets Turing Reducible to P-Selective Sets

95-10 *Boris Hartmann*
Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper

95-12 *Klaus Achatz, Wolfram Schulte*
Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists

95-13 *Andrea Mößle, Heiko Vogler*
Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

95-14 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
A Generic Specification for Verifying Peephole Optimizations

96-01 *Ercüment Canver, Jan-Tecker Gayen, Adam Moik*
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

96-02 *Bernhard Nebel*
Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class

96-03 *Ton Vullinghs, Wolfram Schulte, Thilo Schwinn*
An Introduction to TkGofer

96-04 *Thomas Beuter, Peter Dadam*
Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering

96-05 *Gerhard Schellhorn, Wolfgang Ahrendt*
Verification of a Prolog Compiler - First Steps with KIV

96-06 *Manindra Agrawal, Thomas Thierauf*
Satisfiability Problems

96-07 *Vikraman Arvind, Jacobo Torán*
A nonadaptive NC Checker for Permutation Group Intersection

96-08 *David Cyrluk, Oliver Möller, Harald Rueß*
An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction

96-09 *Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte*
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

96-10 *Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Formalizing Fixed-Point Theory in PVS

96-11 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Mechanized Semantics of Simple Imperative Programming Constructs

96-12 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Generic Compilation Schemes for Simple Programming Constructs

96-13 *Klaus Achatz, Helmuth Partsch*
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

97-01 *Jochen Messner*
Pattern Matching in Trace Monoids

97-02 *Wolfgang Lindner, Rainer Schuler*
A Small Span Theorem within P

97-03 *Thomas Bauer, Peter Dadam*
A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration

97-04 *Christian Heinlein, Peter Dadam*
Interaction Expressions - A Powerful Formalism for Describing Inter-Workflow Dependencies

97-05 *Vikraman Arvind, Johannes Köbler*
On Pseudorandomness and Resource-Bounded Measure

97-06 *Gerhard Partsch*
Punkt-zu-Punkt- und Mehrpunkt-basierende LAN-Integrationsstrategien für den digitalen Mobilfunkstandard DECT

97-07 *Manfred Reichert, Peter Dadam*
$ADEPT_{flex}$ - Supporting Dynamic Changes of Workflows Without Loosing Control

97-08 *Hans Braxmeier, Dietmar Ernst, Andrea Mößle, Heiko Vogler*
The Project NoName - A functional programming language with its development environment

97-09 *Christian Heinlein*
Grundlagen von Interaktionsausdrücken

97-10 *Christian Heinlein*
Graphische Repräsentation von Interaktionsausdrücken

97-11 *Christian Heinlein*
Sprachtheoretische Semantik von Interaktionsausdrücken

97-12 *Gerhard Schellhorn, Wolfgang Reif*
Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers

97-13 *Dietmar Ernst, Frank Houdek, Wolfram Schulte, Thilo Schwinn*
Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme

97-14 *Wolfgang Reif, Gerhard Schellhorn*
Theorem Proving in Large Theories

97-15 *Thomas Wennekers*
Asymptotik rekurrenter neuronaler Netze mit zufälligen Kopplungen

97-16 *Peter Dadam, Klaus Kuhn, Manfred Reichert*
Clinical Workflows - The Killer Application for Process-oriented Information Systems?

97-17 *Mohammad Ali Livani, Jörg Kaiser*
EDF Consensus on CAN Bus Access in Dynamic Real-Time Applications

97-18 *Johannes Köbler,Rainer Schuler*
Using Efficient Average-Case Algorithms to Collapse Worst-Case Complexity Classes

98-01 *Daniela Damm, Lutz Claes, Friedrich W. von Henke, Alexander Seitz, Adelinde Uhrmacher, Steffen Wolf*
Ein fallbasiertes System für die Interpretation von Literatur zur Knochenheilung

98-02 *Thomas Bauer, Peter Dadam*
Architekturen für skalierbare Workflow-Management-Systeme - Klassifikation und Analyse

98-03 *Marko Luther, Martin Strecker*
A guided tour through *Typelab*

98-04 *Heiko Neumann, Luiz Pessoa*
Visual Filling-in and Surface Property Reconstruction

98-05 *Ercüment Canver*
Formal Verification of a Coordinated Atomic Action Based Design

98-06 *Andreas Küchler*
On the Correspondence between Neural Folding Architectures and Tree Automata

98-07 *Heiko Neumann, Thorsten Hansen, Luiz Pessoa*
Interaction of ON and OFF Pathways for Visual Contrast Measurement

98-08 *Thomas Wennekers*
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

98-09 *Thomas Bauer, Peter Dadam*
Variable Migration von Workflows in *ADEPT*

98-10 *Heiko Neumann, Wolfgang Sepp*
Recurrent V1 – V2 Interaction in Early Visual Boundary Processing

98-11 *Frank Houdek, Dietmar Ernst, Thilo Schwinn*
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment

98-12 *Gerhard Schellhorn*
Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers

98-13 *Gerhard Schellhorn, Wolfgang Reif*
Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers

98-14 *Mohammad Ali Livani*
SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN

98-15 *Mohammad Ali Livani, Jörg Kaiser*
Predictable Atomic Multicast in the Controller Area Network (CAN)

99-01 *Susanne Boll, Wolfgang Klas, Utz Westermann*
A Comparison of Multimedia Document Models Concerning Advanced Requirements

99-02 *Thomas Bauer, Peter Dadam*
Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation

99-03 *Uwe Schöning*
On the Complexity of Constraint Satisfaction

99-04 *Ercument Canver*
Model-Checking zur Analyse von Message Sequence Charts über Statecharts

99-05 *Johannes Köbler, Wolfgang Lindner, Rainer Schuler*
Derandomizing RP if Boolean Circuits are not Learnable

99-06 *Utz Westermann, Wolfgang Klas*
Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets

99-07 *Peter Dadam, Manfred Reichert*
Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. Paderborn, Germany, October 6, 1999, GI–Workshop Proceedings, Informatik ’99

99-08 *Vikraman Arvind, Johannes Köbler*
Graph Isomorphism is Low for ZPP^{NP} and other Lowness results

99-09 *Thomas Bauer, Peter Dadam*
Efficient Distributed Workflow Management Based on Variable Server Assignments

2000-02 *Thomas Bauer, Peter Dadam*
Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

2000-03 *Gregory Baratoff, Christian Toepfer, Heiko Neumann*
Combined space-variant maps for optical flow based navigation

2000-04 *Wolfgang Gehring*
Ein Rahmenwerk zur Einführung von Leistungspunktsystemen

2000-05 *Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel*
Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos

2000-06 *Wolfgang Reif, Gerhard Schellhorn, Andreas Thums*
Fehlersuche in Formalen Spezifikationen

2000-07 *Gerhard Schellhorn, Wolfgang Reif (eds.)*
FM-Tools 2000: The 4th Workshop on Tools for System Design and Verification

2000-08 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen

2000-09 *Thomas Bauer, Peter Dadam*
Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT

2000-10 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Adaptives und verteiltes Workflow-Management

2000-11 *Christian Heinlein*
Workflow and Process Synchronization with Interaction Expressions and Graphs

2001-01 *Hubert Hug, Rainer Schuler*
DNA-based parallel computation of simple arithmetic

2001-02 *Friedhelm Schwenker, Hans A. Kestler, Günther Palm*
3-D Visual Object Classification with Hierarchical Radial Basis Function Networks

2001-03 *Hans A. Kestler, Friedhelm Schwenker, Günther Palm*
RBF network classification of ECGs as a potential marker for sudden cardiac death

2001-04 *Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm*
Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion

2002-01 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata

2002-02 *Walter Guttmann*
Deriving an Applicative Heapsort Algorithm

2002-03 *Axel Dold, Friedrich W. von Henke, Vincent Vialard, Wolfgang Goerigk*
A Mechanically Verified Compiling Specification for a Realistic Compiler

2003-01 *Manfred Reichert, Stefanie Rinderle, Peter Dadam*
A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks

2003-02 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Supporting Workflow Schema Evolution By Efficient Compliance Checks

2003-03 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values

2003-04 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
On Dealing With Semantically Conflicting Business Process Changes.

2003-05 *Christian Heinlein*
Dynamic Class Methods in Java

2003-06 *Christian Heinlein*
Vertical, Horizontal, and Behavioural Extensibility of Software Systems

2003-07 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values
(Corrected Version)

2003-08 *Changling Liu, Jörg Kaiser*
Survey of Mobile Ad Hoc Network Routing Protocols)

2004-01 *Thom Frühwirth, Marc Meister (eds.)*
First Workshop on Constraint Handling Rules

2004-02 *Christian Heinlein*
Concept and Implementation of C+++, an Extension of C++ to Support User-Defined Operator Symbols and Control Structures

2004-03 *Susanne Biundo, Thom Frühwirth, Günther Palm(eds.)*
Poster Proceedings of the 27th Annual German Conference on Artificial Intelligence

2005-01 *Armin Wolf, Thom Frühwirth, Marc Meister (eds.)*
19th Workshop on (Constraint) Logic Programming

2005-02 *Wolfgang Lindner (Hg.), Universität Ulm , Christopher Wolf (Hg.) KU Leuven*
2. Krypto-Tag – Workshop über Kryptographie, Universität Ulm

2005-03 *Walter Guttmann, Markus Maucher*
Constrained Ordering

2006-01 *Stefan Sarstedt*
Model-Driven Development with ACTIVECHARTS, Tutorial

2006-02 *Alexander Raschke, Ramin Tavakoli Kolagari*
Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethode zur Spezifikation von eingebetteten Systemen

2006-03 *Jens Kohlmeyer, Alexander Raschke, Ramin Tavakoli Kolagari*
Eine qualitative Untersuchung zur Produktlinien-Integration über Organisationsgrenzen hinweg

2006-04 *Thorsten Liebig*
Reasoning with OWL - System Support and Insights –

2008-01 *H.A. Kestler, J. Messner, A. Müller, R. Schuler*
On the complexity of intersecting multiple circles for graphical display

2008-02 *Manfred Reichert, Peter Dadam, Martin Jurisch,l Ulrich Kreher, Kevin Göser, Markus Lauer*
Architectural Design of Flexible Process Management Technology

2008-03 *Frank Raiser*
Semi-Automatic Generation of CHR Solvers from Global Constraint Automata

2008-04 *Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander*
Entscheidungsdokumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse

2008-05 *Markus Kalb, Claudia Dittrich, Peter Dadam*
Support of Relationships Among Moving Objects on Networks

2008-06 *Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)*
WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke

2008-07 *M. Maucher, U. Schöning, H.A. Kestler*
An empirical assessment of local and population based search methods with different degrees of pseudorandomness

2008-08 *Henning Wunderlich*
Covers have structure

2008-09 *Karl-Heinz Niggl, Henning Wunderlich*
Implicit characterization of FPTIME and NC revisited

2008-10 *Henning Wunderlich*
On span-P^{cc} and related classes in structural communication complexity

2008-11 *M. Maucher, U. Schöning, H.A. Kestler*
On the different notions of pseudorandomness

2008-12 *Henning Wunderlich*
On Toda's Theorem in structural communication complexity

2008-13 *Manfred Reichert, Peter Dadam*
Realizing Adaptive Process-aware Information Systems with ADEPT2

2009-01 *Peter Dadam, Manfred Reichert*
The ADEPT Project: A Decade of Research and Development for Robust and Fexible Process Support
Challenges and Achievements

2009-02 *Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, Martin Jurisch*
Von ADEPT zur AristaFlow® BPM Suite – Eine Vision wird Realität "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen

2009-03 *Alena Hallerbach, Thomas Bauer, Manfred Reichert*
Correct Configuration of Process Variants in Provop

2009-04 *Martin Bader*
On Reversal and Transposition Medians

2009-05 *Barbara Weber, Andreas Lanz, Manfred Reichert*
Time Patterns for Process-aware Information Systems: A Pattern-based Analysis

2009-06 *Stefanie Rinderle-Ma, Manfred Reichert*
Adjustment Strategies for Non-Compliant Process Instances

2009-07 *H.A. Kestler, B. Lausen, H. Binder H.-P. Klenk. F. Leisch, M. Schmid*
Statistical Computing 2009 – Abstracts der 41. Arbeitstagung

2009-08 *Ulrich Kreher, Manfred Reichert, Stefanie Rinderle-Ma, Peter Dadam*
Effiziente Repräsentation von Vorlagen- und Instanzdaten in Prozess-Management-Systemen

2009-09 *Dammertz, Holger, Alexander Keller, Hendrik P.A. Lensch*
Progressive Point-Light-Based Global Illumination

2009-10 *Dao Zhou, Christoph Müssel, Ludwig Lausser, Martin Hopfensitz, Michael Kühl, Hans A. Kestler*
Boolean networks for modeling and analysis of gene regulation

2009-11 *J. Hanika, H.P.A. Lensch, A. Keller*
Two-Level Ray Tracing with Recordering for Highly Complex Scenes

2009-12 *Stephan Buchwald, Thomas Bauer, Manfred Reichert*
Durchgängige Modellierung von Geschäftsprozessen durch Einführung eines Abbildungsmodells: Ansätze, Konzepte, Notationen

2010-01 *Hariolf Betz, Frank Raiser, Thom Frühwirth*
A Complete and Terminating Execution Model for Constraint Handling Rules

2010-02 *Ulrich Kreher, Manfred Reichert*
Speichereffiziente Repräsentation instanzspezifischer Änderungen in Prozess-Management-Systemen

2010-03 *Patrick Frey*
Case Study: Engine Control Application

2010-04 *Matthias Lohrmann und Manfred Reichert*
Basic Considerations on Business Process Quality

2010-05 *HA Kestler, H Binder, B Lausen, H-P Klenk, M Schmid, F Leisch (eds):*
Statistical Computing 2010 - Abstracts der 42. Arbeitstagung

2010-06 *Vera Künzle, Barbara Weber, Manfred Reichert*
Object-aware Business Processes: Properties, Requirements, Existing Approaches

2011-01 *Stephan Buchwald, Thomas Bauer, Manfred Reichert*
Flexibilisierung Service-orientierter Architekturen

2011-02 *Johannes Hanika, Holger Dammertz, Hendrik Lensch*
Edge-Optimized À-Trous Wavelets for Local Contrast Enhancement with Robust Denoising

2011-03 *Stefanie Kaiser, Manfred Reichert*
Datenflussvarianten in Prozessmodellen: Szenarien, Herausforderungen, Ansätze

2011-04 *Hans A. Kestler, Harald Binder, Matthias Schmid, Friedrich Leisch, Johann M. Kraus (eds):*
Statistical Computing 2011 - Abstracts der 43. Arbeitstagung

2011-05 *Vera Künzle, Manfred Reichert*
PHILharmonicFlows: Research and Design Methodology

2011-06 *David Knuplesch, Manfred Reichert*
Ensuring Business Process Compliance Along the Process Life Cycle

2011-07 *Dausend, Marcel*
Towards a UML Profile on Formal Semantics for Modeling Multimodal Interactive Systems

2011-08 *Gessenharter, Dominik*
Model-Driven Software Development with ACTIVECHARTS - A Case Study

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm