Faculty of Mathematics and Physics, Charles University in Prague
Department of Software Engineering

Use Cases: Behavior Assembly, Behavior Composition and Reasoning

Doctoral Thesis Abstract

Vladimír Mencl

Advisor: Prof. Ing. František Plášil, DrSc.
Faculty of Mathematics and Physics, Charles University in Prague
Department of Software Engineering

I2 Software systems

Prague, June 2004

Matematicko-fyzikální fakulta, Univerzita Karlova v Praze
Katedra softwarového inženýrství

Use Cases: Behavior Assembly, Behavior Composition and Reasoning

Autoreferát doktorské disertační práce

Vladimír Mencl

Školitel: Prof. Ing. František Plášil, DrSc.
Matematicko-fyzikální fakulta, Univerzita Karlova v Praze
Katedra softwarového inženýrství

I2 Softwarové systémy

Praha, červen 2004

Tato disertační práce byla vypracována v rámci doktorského studia na Matematicko-fyzikální fakultě Univerzity Karlovy v Praze v letech 1998-2004.

Doktorand: Mgr. Vladimír Mencl

Školitel: Prof. Ing. František Plášil, DrSc.
Katedra softwarového inženýrství
MFF UK Praha

Školící pracoviště:
Katedra softwarového inženýrství
MFF UK Praha
Malostranské náměstí 25
118 00 Praha 1

Oponenti:

(1) Dr. Martin Große-Rhode
Fraunhofer-ISST, Mollstr. 1
10178 Berlin, Germany
tel.: +49(0)30/24306-353
email: Martin.Grosse-Rhode@isst.fraunhofer.de

(2) RNDr. Zdenko Staníček, Ph.D.
Fakulta informatiky
Katedra programových systémů a komunikací
Botanická 68a
602 00 Brno
tel.: +420 549 496 993
email: stanicek@informatics.muni.cz

Autoreferát byl rozeslán dne ....

Obhajoba se koná dne .... v .... hodin před komisí pro obhajoby disertačních prací v oboru I-2 Softwarové systémy na Matematicko-fyzikální fakultě Univerzity Karlovy v Praze, Ke Karlovu 3, 121 16 Praha 2 v místnosti č. 105.

S disertací je možno se seznámit na studijním oddělení doktorského studia Matematicko-fyzikální fakulty Univerzity Karlovy, Ke Karlovu 3, Praha.

Předseda RDSO I-2: Prof. Ing. František Plášil, DrSc.

Contents

Contents 4

1. Introduction 5
   1.1. Software Components and Behavior Specifications 5
   1.2. Use Case Modeling 6
   1.3. Assembly and Composition 7
   1.4. UML 2.0 8
   1.5. Problem Statement 8
   1.6. Goal of the Thesis 9

2. Key Results 10
   2.1. Use Cases Elaborated: Generic UC View 10
      2.1.1. Consistency Issues 11
   2.2. Analysis Summary: Textual Use Cases and UML 1.x 12
   2.3. Pro-cases 13
   2.4. Converting Textual Use Cases into Pro-cases 15
   2.5. Analyzing Use Cases in UML 2.0 17
      2.5.1. Basic UC View 17
      2.5.2. Trace-Based UC View 18
      2.5.3. Analyzing UML 2.0: Assembled and Composed Behavior 19
   2.6. Port State Machines 19

3. Summary of Contribution 21
   3.1. Summary 21
   3.2. Contribution 22

References 23

Author's Publications 26

1. Introduction

1.1. Software Components and Behavior Specifications

The prevailing trend in software engineering is to design software systems by composition of *software components* [21]. Although there are many views on what constitutes a software component, the consensus is that a component is characterized by the services it provides and requires, typically via interfaces; the component models Darwin [23], Wright [2], SOFA [40] and Fractal [5] follow this paradigm. In addition, a component may be composed of nested (sub-)components; SOFA [40] and Fractal [5] are examples of such hierarchical component models.

The interfaces to services are described in terms of signatures of operations available on the interface. This allows to reason on correctness of component composition at the level of syntactical type correspondence. However, when considering composition of components obtained from different sources (vendors, development teams), such a specification is clearly not sufficient. A specification of the component's behavior is essential, describing the correct usage of the services provided by the component, the intended use of the services required by the component, as well as the dependence among interactions occurring on the individual interfaces of the component.

Often, behavior is specified only in an informal (plain language) description, in the form of additional documentation. Meanwhile, research in the field of formal methods has achieved a state where specification methods are available to specify the behavior of a component in a way that is (i) easy to learn, (ii) employs a human-readable notation, and (iii) provides means to reason on correctness of composition. An example of such a behavior specification method are Behavior Protocols [44] employed in the SOFA component model [40]. Besides textual notations, visual notations such as message sequence charts (MSC) and State-charts [13] and are used; here, verification methods employ generating test scenarios via simulation [9, 12], as well as translating the (complex) specification into a labeled transition system (LTS) as the common denominator and reasoning on the LTS [20].

Applying behavior specification methods to Component Based Software Development (CBSD) can yield significant benefits, either as the option to employ a tool to verify the specification of the system being designed, or the possible CASE support in developing the component implementation. A CASE tool may utilize the behavior specification in deriving the initial design, as well as in generating skeleton code for operation implementations.

As use cases may be used as the starting point in specifying the behavior of a system or a component, employing use cases specifications to achieve these benefits would be very convenient.

1.2. Use Case Modeling

In principle a use case [16, 7, 37] is a description of a set of scenarios specifying how a set S of entities ought to communicate to achieve a certain goal; a communication is viewed as a sequence of events, such as a request or a response, exchanged among the entities. Here, an entity can represent a system, a subsystem, an agent or a software component. Frequently, the use case is written from the perspective of one of those entities (*SuD*, system under discussion) – it specifies how SuD executes certain actions while communicating with other entities, *actors,* from S to achieve a specific goal. Basically, a *scenario* is considered to be a sequence of *actions* to be performed by SuD and the actors, which reflects a particular case of their desired communication.

Use Case: MIS#1 Seller submits an offer
Scope: Marketplace
SuD: Marketplace Information System
Level: Primary Task
Primary Actor: Seller
Supporting Actor: Trade Commission
Main success scenario specification:
1. Seller submits item description
2. System validates the description.
3. Seller adjusts/enters price and enters contact and billing information.
4. System validates the seller's contact information.
5. System verifies the seller's history to permit the seller to operate
6. System validates the whole offer with the Trade Commission
7. System lists the offer in published offers.
8. System responds with a uniquely identified authorization number.

Extensions:
2a Item not valid
    2a1 Use case aborted
5a Seller's history inappropriate
    5a1 Use case aborted
6a Trade commission rejects the offer
    6a1 Use case aborted
Sub-variations:
2b Price assessment available
    2b1 System provides the seller with a price assessment.

Figure 1 Sample Textual Use Case

Practitioners, e.g. [7, 19, 39, 48], typically prefer use cases to be specified in plain English, to make them easily comprehendible to "wide audience". Such a use case is inherently informal, even though a predefined template is usually asked to follow. For example, such a template can be a form to be filled in to specify in a semi-programming way the desired set of scenarios (fig. 1). There is a whole variety of ways different authors recommend to write use cases, ranging, e.g., from employing preconditions/postconditions in Catalysis [10], Use Case Maps [3], transition systems [50], to abstract state machines with the goal to generate test scenarios [12].

UML [37] includes a use case concept as well. It is, however, primarily focused on use case as an abstraction to capture the existence of a set of interaction scenarios among a set of actors and an SuD; it leaves the way internals of a use case are specified very open (the alternatives explicitly mentioned without any details include plain text, a state machine, activity graph, and specification via preconditions and postconditions). Thus, UML rather concentrates on the relations among use cases.

In general, the bottom line is that there are many different approaches and hard to compare techniques related to use cases, none of them being strongly recommended nor preferred; an overview is, e.g., in [15, 11].

Intuitively, there can be conflicts in use cases specifying two cooperating entities (separate SuDs). Even though there are many approaches to finding conflicts in dynamic and functional requirements, as pointed out in [14], they are frequently based on logic (and typically closely dependent on a particular use case technology) and require highly specialized experts to handle. This is in obvious contrast with practitioners' desire to make a use case easy to read and comprehend as mentioned above.

1.3. Assembly and Composition

We illustrate the issues of assembly and composition on a sample marketplace application. The scope diagram in Fig. 2 shows all the entities involved in this example, their nesting and communication links. Figure 1 shows a sample textual use case of the entity Marketplace Information System (MIS). Apparently, there would be more use cases of MIS – all of them together specifying the behavior of MIS. The scenarios specified by these individual use cases may be required to be performed in a sequence, concurrently, etc.; in general we call the process of "putting use cases together" *behavior assembly* which yields the *assembled behavior* of a SuD. In this view, the *use case model* of MIS is determined by all its use cases and by the way they are combined.

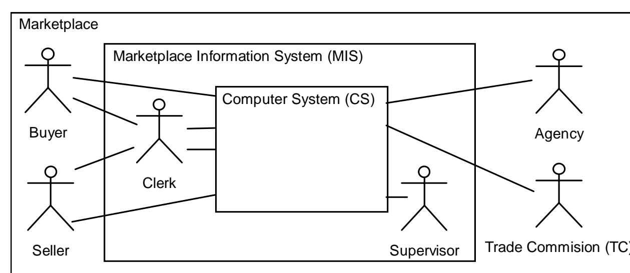

Figure 2: Scope diagram of the Marketplace system

In a similar vein, for composed entities the question is whether it is possible to obtain the behavior of the containing entity by *composition* of the assembled behaviors of the nested entities. The *composed behavior* has to capture the internal communication among the nested entities, as well as those of their activities that are

visible outside as the behavior of the containing entity. In the example above, MIS communicates with Trade Commission (TC); in a possible use case of TC, several steps would correspond to the step 6 of MIS#1. Except for this special case (synchronization), in which the complementary communication actions become an internal activity, these entities operate in parallel and their activities interleave.

If composition semantics was precisely defined for a specification mechanism, it might be possible to reason on behavior consistency of nested entities, e.g., whether the composed behavior of Clerk, Computer System and Supervisor is consistent with the assembled behavior specified for MIS.

1.4. UML 2.0

While UML 1.5 is still the current official industry standard, the emerging standard UML 2.0 is reaching completion. UML 2.0 features four behavior specification mechanisms that may be used for use cases: *Interactions*, *Activities*, *State Machines* and *Protocol State Machines* (PSM). Moreover, UML 2.0 introduces the concepts *StructuredClassifier* and *EncapsulatedClassifier*, providing support for modeling internal structure and featuring *Port*s; a *Port* is associated with a set of *provided* and *required* interfaces. Based on these concepts, a *Component* may be captured in a UML model; nesting is supported and the external communication of the component is encapsulated in the component's *Ports*.

As for behavior reasoning, UML explicitly considers *"conformance"* of PSMs; however, the role of conformance is limited to explicitly declaring that a *specific* StateMachine (possibly a PSM) conforms to a *general* PSM. Note that UML defines the semantics of protocol conformance only partially and it is not clear under which circumstances protocol conformance may be declared and thus, it is not feasible to automatically decide on protocol conformance. UML employs the protocol conformance in the *Components* framework, requiring *realization* of a *Component* (possibly a StateMachine specifying the component) to be conforming with specifications of all its *Interfaces*.

1.5. Problem Statement

Typically, use cases are written only informally, as plain language descriptions. In CBSD, employing formal methods in use case specifications might provide beneficial results. Capturing the "whole picture" of the behavior of an entity (a component) may be used by a tool to aid in creating the design of the component's implementation. Moreover, when supported by the formalism employed, behavior reasoning may be used to validate the composition of components in the system under development.

When use cases are used to capture the requirements of a future system, the behavior specification is scattered across the set of use cases. Although several authors propose formal methods for specifying use case behavior, the issues of obtaining the "whole picture" of an entity's behavior, behavior assembly and

behavior composition have not been explicitly considered for use cases so far, neglecting the possible benefits. When permitted by the formalism employed, both the design and validation features might be implemented in a tool, possibly as a plugin to a CASE (UML) environment.

1.6. Goal of the Thesis

The goals of the thesis are to explore the relations among use cases employed to specify behavior of a software system. The initial goal (i) is to analyze and formally define the concepts of behavior assembly and behavior composition. Here, a supplementary goal (ii) is to develop a formal model to capture these concepts; this model should also serve as the basis for consistency reasoning.

The goal (iii) is to use the formal model to analyze how the concepts identified are reflected in traditional use case approaches. The analysis should cover at least the textual use cases [7] and the support of UML 1.4/1.5 [36, 37] for use cases. The analysis should examine:

1. how are the basic abstractions interpreted (entity, use case)
2. whether and how the assembled behavior (representing “whole picture” of the behavior) of an entity can be obtained.
3. how the consistency issues are addressed.

The goal (iv) is to utilize the results of these analyses and propose a use case technology that provides an interpretation for the abstractions defined by the model; the proposed technology should support behavior assembly, behavior composition and consistency reasoning; in consistency reasoning, the relations addressing consistency reasoning should be decidable in a computationally feasible way. A related goal (v) is to propose conversion from the textual use cases to this notation; to prove the feasibility of the proposal, another part of this goal is to implement a tool transforming textual use cases into this formal notation.

Afterwards, the goal (vi) is to focus on the upcoming standard UML 2.0 and analyze the behavior specification mechanisms it provides. Here, in order to handle the diversity of the specification mechanisms available in UML 2.0, an additional task may be to extend the formal model in a way that permits analyzing these specification mechanisms defined at varying levels of clarity and unambiguity.

Consequently, the goal (vii) is to employ the analysis results and propose a UML specification mechanism (proposed as an extension to UML 2.0) that supports consistency reasoning, as well as behavior composition, behavior assembly, and interpreting the assembly operations via native operations of the specification mechanism.

2. Key Results

2.1. Use Cases Elaborated: Generic UC View

In [44, 45], our research group developed an agent model, where the agents process sequences (traces) of atomic *events*, and introduced a way to describe (approximate) the agents' behavior via behavior protocols, which was applied on software components in SOFA in order to specify component behavior and test behavior compliance of components, including neighboring levels of nesting. Based on this experience, we realized that similar compliance checks should be done also for use case models associated with component-based (interface-centered [10]) design. To provide a basis for reasoning about the key abstractions (and capture their relationship) in the traditional use case modeling [7, 16, 17, 37], we introduce the following generic model (*Generic UC View*).

Basic concepts. Assume an entity S is composed of sub-entities A_1, ... ,A_n. By definition S forms the *scope* of A_i; the topmost scope is called *system*. An entity A_i communicates through communication links (*connections* for short) with (1) other (*actors*) A_j of the scope S, and potentially (2) with other external actors located in the parent scope, i.e., in the scope of S. In case (1), the communication is observed on the *internal* connections of S, while in case (2) on *external* connection of S. Advantageously, the nesting of entities and their scopes can be expressed as a scope diagram (fig. 2). Note that a scope diagram captures the relations among entities, not among use cases and entities (as UML use case diagram does).

Scenarios. A particular way of communication of an entity A on its connections in a run of system Σ is captured as a *scenario* s $\in$ Scenarios. All the scenarios of A in any run of Σ form the *behavior* Com(A) $\subseteq$ Scenarios. On the domain Scenarios we assume the existence of a subscenario relation (partial order).

By convention, Com(A)/ExConn(A) is the behavior of A restricted to its external connections (while Com(A)/InConn(A) is restricted to the internal connections of A); here, Com(A)/Conn denotes restriction of scenarios from Com(A) to the communication observed on the connections from a set of connections Conn.

Assuming an entity C is composed of entities A and B, Com(C) is composed of Com(A) and Com(B), written Com(A) $\sqcap_X$ Com(B), where X = ExConn(A) $\cap$ ExConn(B), in such a way that

(1) together with Com(A) / InConn(A) $\cup$ Com(B) / InConn(B) the behavior on the joint connections between A and B becomes Com(C) / InConn(C),

(2) (Com(A) / ExConn(A) $\cup$ Com(B) / ExConn(B)) –
(Com(A) / ExConn(A) $\cap$ Com(B) / ExConn(B))
becomes Com(C)/ExConn(C) (where – stands for set subtraction)

Use case model. Let U^A be the set of *basic use cases* (behavior specifications) where A is the SuD. A use case $UC^A_i \in U^A$ describes/generates a set of scenarios; so, by convention, we write also Com(UC^A_i) $\subseteq$ Scenarios. Further, we define a binary relation includes such that UC^A_j includes UC^A_k means that the specification of UC^A_j includes (refers to) the specification of UC^A_k (macro substitution idea, thus no circular dependencies allowed). Also we assume that if $UC^A_j \in U^A$ and UC^A_j includes

UC^A_k then also $UC^A_k \in U^A$; moreover we require UC^A_j <u>includes</u> UC^A_k to imply that for any s $\in$ Com(UC^A_k) there exists an s' $\in$ Com(UC^A_j) such that s <u>subscenario</u> s' (*subscenario preservation*).

Notice that even if Com(UC^A_k) = Com(UC^A_j), not necessarily $UC^A_k = UC^A_j$; thus different specifications can generate the same behavior. Also, in the need to distinguish the elements of U^A we do so by subscripts, writing, e.g., UC^A_k (this also reflect that use cases of A are enumerated in a typical use case technology).

A *use case model* of A (denoted UM^A) is a set of *use case expressions*, (also *use cases* for short), where an expression UE^A (syntactically in principle) generates a set of scenarios (by convention we write Com(UE^A) $\subseteq$ Scenarios). A use case expression UE^A is either a basic use case UC^A_i or is composed by applying operations from a set of operations UEop on their operands – (sub)expressions, assuming some priorities, parenthesis, etc. apply. The semantic of these operations can include sequencing, parallel composition of the scenarios generated by the operands, etc. The <u>includes</u> relation can be naturally extended to use case expressions.

Whole picture behavior. As a use case $UE^A_j \in UM^A$ provides only a partial "j-th" description of A's behavior ("the whole picture behavior" of A),the *assembled behavior* of UM^A is defined as Com(UM^A) = $\cup_j$ Com(UE^A_j), $UE^A_j \in UM^A$. To emphasize an important special case, we say that UM^A *has a representative* if there is UE^A in UM^A such that Com(UM^A) = Com(UE^A); (also: UE^A is an *representative* of UM^A).

2.1.1. Consistency Issues

Inherent to refinement/synthesis steps in a design of a specified system Σ is the necessity to combine behavior specifications in order to get "the whole picture" behavior specification and capture the behavior specification compliance of several cooperating/nested entities. In particular, the following four issues are closely related to this necessity:

(a) Does the combined behavior, as specified by all UM^{Ai} in a scope S, comply with the behavior specified for S in UM^S.

(b) Does the assembled behavior, as specified by a UM^A, really reflect the desired behavior of A (as this is hard to address directly, we will consider equivalence checking − decidability whether two use case models UM^A and 'UM^A specify the same behavior, i.e., Com(UM^A) = Com('UM^A)).

(c) Is the desired communication between A_i and A_j via their connection(s) in S really reflected in the behavior as specified (separately) by UM^{Ai} and UM^{Aj}.

(d) If there is no representative of UM^A and the behaviors of use cases in UM^A overlap, is there a way to find/construct a representative in order to get a "whole picture behavior" directly from the specifications, without the need to generate scenarios.

In general, addressing (a) and (b) requires defining a *behavior compliance* as a binary relation upon the behavior of entities. Intuitively, Com(A) <u>compliant with</u> Com(B) if B can be replaced in Σ by A by taking over all its external connections in such a way that "B behaves in A's place as it were A". The issue (c) can be addressed

by finding a binary relation <u>consent</u> upon behavior of entities: Intuitively, Com(A) <u>consent</u> Com(B) if there is no inconsistency resp. "erroneous scenario" in the behavior of A and B on their joint connections. To define these relations (and $\sqcap_X$) precisely, a specific interpretation of Scenarios and the relations/operations available for it have to be known.

Note that the relations are (intentionally) defined only for Com(A) and Com(B) and not A and B, as in a concrete UC view, behavior of A will be approximated by Com(UM^A); in Pro-cases, the language generated by a protocol approximates the behavior of an entity.

Assuming the existence of <u>compliant with</u> and <u>consent</u>, the issues (a) - (c) can be rephrased as

(a) Com(UM^{A1}) $\sqcap_{X1}$ Com(UM^{A2}) $\sqcap_{X2}$ .... $\sqcap_{Xn-1}$ Com(UM^{An}) <u>compliant with</u> Com(UM^S), (here X_i is ExConn(A_i) $\cap$ ExConn(A_{i+1}))

(b) Com(UM^A) <u>compliant with</u> Com('UM^A) and Com('UM^A) <u>compliant with</u> Com(UM^A)

(c) Com(A_i) <u>consent</u> Com(A_j)

Obviously a big advantage can be taken of extending the definitions of <u>compliant with</u>, <u>consent</u> and $\sqcap_X$ to UM^A (to make them applicable not only on Com(A), i.e. the behavior itself but also on the behavior specification). Then, assuming a set UEop is "reasonably" defined, the consistency issues (a) - (d) can be addressed by reasoning on uses case expression (as in the case of the specification mechanism Pro-cases we propose). An example of defining the <u>compliant with</u> and <u>consent</u> relations is, besides this thesis, available in [1, 44].

2.2. Analysis Summary: Textual Use Cases and UML 1.x

We have analyzed the textual use cases and the support for use cases in UML 1.4/1.5. The basic concepts (entity, SuD, actor, connection) are reflected in both of these technologies. However, none of them explicitly defines the domain <u>*Scenarios*</u> (although for textual use cases, <u>*Scenarios*</u> is implicitly assumed). Consequently, composition is not defined for UML 1.x and is difficult to interpret for textual use cases.

No operations for assembling behavior are defined for UML 1.x. For textual use cases, sequencing, repetition and alternative can be used; semantics of parallel composition cannot be reasonably defined. The traditionally used summary use case addresses only the simple case of exclusively using the sequencing operation.

Both the technologies have the notion of a use case model, however, the information for constructing the whole picture behavior is not captured in UML, and only in a limited form in textual use cases.

With no concrete specification mechanism prescribed, the issue (d) of constructing a representative use case is not addressed in UML 1.x. In textual use case, a use case may be constructed for use case expressions employing only the operations sequencing, repetition and alternative, however, for complex expressions, the

resulting use case would be very hard to manage; in the traditional textual use case recommendations, nested extensions are discouraged.

As no scenarios domain is provided (and as the composition operation cannot be interpreted), consistency reasoning is not addressed in neither of these technologies.

2.3. Pro-cases

We proposed *Pro-cases* (short for *Protocol use cases*), a specification mechanism based on Behavior Protocols [44, 45], developed within our group. Pro-cases, a concrete instance of our Generic UC View, feature all the relations asked for in Sect. 2.1.1, providing a proof of the Generic UC View concept; in addition, the compliance relation is decidable and a verifier tool is available [22, 49].

In [44] behavior of an entity is modeled as a set of traces (finite sequences of atomic events), capturing the communication on the entity's connections. A regular expression-like notation is used to approximate the actual behavior of entities by regular languages; the notation employs the operators: ***** (repetition), **;** (sequencing), **+** (alternative), **|** (parallelism), **||** (parallel-or, shortcut for A + B + A|B) and the composed operators *composition* ($\sqcap_X$), *adjustment* ($|_X|$) and *consent* ($\triangledown_X$). Behavior protocols are employed in the SOFA hierarchical component model [40, 44]. As the behavior is formally defined and there are powerful operations upon the protocols and behavior (languages), decidable relations and operations exist ($\sqcap_X$, compliant with, consent [1]) which allow to decide on compatibility of two components (their specifications). We show, that (and how) the behavior protocol concept fits into the generic UC view (a key idea here is that a "use case" corresponds to a "protocol").

Basic concepts, Scenarios. An agent corresponds to the entity concept; a scenario (called a *trace* in behavior protocols) is a finite sequence of atomic events. The events are denoted by event tokens from a domain ACT in [44]. For our purpose, we assume ACT* corresponds to the Scenarios domain. The subscenario relation is thus defined via a subsequence relation (employing correspondence of !/? to internal events τ) and is therefore decidable. An event is modeled as an event token *a* either emitted (*!a),* absorbed (*?a)* or internally processed (τa). *Com*(A) $\subseteq$ ACT* denotes the behavior of an entity A (a language upon ACT).

The following example suggests (in a simplified form), how the scenario generated by a use case could be mapped to a trace of a behavior protocol.

<?sic.submitItem, τValidateItem, ?sic.submitPrice, τValidateSeller, τVerifySellerHistory, !tradecom.validate, τListOffer, !sellernotify.putAuthNr>

The trace corresponds to the main success scenario specification of the use case shown in Fig. 1. The actions performed internally by SuD are represented as internal actions (τ), the actions performed by an actor toward SuD are captured as absorbed (by SuD, *?* used), actions performed by SuD toward an actor are captured as emitted (*!*). The event token domain ACT contains names composed of a connection name (e.g., *sic*) and event name (*submitItem*).

Use cases and relations. A *Pro-case* with an entity A as SuD is a behavior protocol $Prot^A$ approximating the behavior of A by bounding the behavior of A. As an example, we show a behavior protocol fragment corresponding to actions 1 and 2 of the sample use case shown in Fig. 1 (the extensions and variations pertaining to these lines are considered). The events corresponding to the main success scenario specification are printed in **bold**.

?sic.submitItem { **τValidateItem** ; (**Null** + τPriceAssessmentAvailable ; !sellernotify.putPriceAssessment + τInvalidItem)
}

The includes relation is defined as an inclusion of behavior protocol specifications (similar to macro substitution, naturally acyclic); the subscenario preservation property is implied from the definition of Pro-cases.

Whole picture behavior. Typically, only a single Pro-case is used (as the representative of UM^A); such a Pro-case is called the *frame Pro-case* (inspired by *frame protocol* in SOFA [44, 40]). The basic and parallel operators used in the behavior protocols notation can be advantageously employed as the operations for use case expressions (the UEop set); thus, assembling the behavior via use case expressions is natural here.

Addressing consistency issues. Even though the variety of the behavior protocol operators provides strong expressive power (strong enough to describe concurrency, procedure calls, etc.), the behavior (language) generated is a regular language. This significant advantage allows for comparing behavior described by behavior protocols, as, e.g., inclusion of regular languages is decidable. The composition operation $\sqcap_X$ conforms to the generic view of composition (from Generic UC View). The relations compliant with and consent are defined and are decidable, thus the consistency issues (a), (b) and (c) are addressed here.

?sic.submitItem { **τValidateItem** ; (NULL +
τPriceAssessmentAvailable ;
!sellernotify.putPriceAssessment +
τInvalidItem) **}** ;
(**?sic.submitPrice {** **τValidateSeller** ;
τVerifySellerHistory ; (
!tradecom.validate ; (**τListOffer** ;
!sellernotify.putAuthNr +
τTradeComValidateFailed) + τVerifyFailed
)
} + τInvalidItem
)

Figure 4: Pro-case manually created for the use case "Seller submits an offer"

2.4. Converting Textual Use Cases into Pro-cases

In [43], we proposed guidelines for transforming textual use cases into Pro-cases by hand; Figure 4 demonstrates the Pro-case manually obtained in [43] for the textual use case shown in Fig. 1. Surprisingly, readily available tools for natural language processing can be employed to accomplish this task in an automated way. We describe how the simple and uniform structure of sentences describing steps of a use case can be utilized to extract the principal attributes of the step's action from its parse tree. We achieve this task with only a minimal domain model.

Readily available linguistic tools permit to acquire a parse tree from a natural language sentence. A *phrase structure* parse tree captures the structure of the sentence according to the grammar of the respective natural language; the leaves reflect the words of the sentence (in left-to-right order), while intermediary nodes represent *phrases* constituting the structure of the sentence. Figure 5 shows a parse tree obtained for the sentence of step 1 of the use case shown in Fig. 1. There, the nouns "item" and "description" constitute a *basic noun phrase* (denoted NPB), which together with the verb "submits" forms a *verb phrase* (VP). The parse tree also shows the headword of each phrase (e.g., the verb "submits" for the verb phrase).

The part-of-speech (POS) determines word type and the role a word plays in the phrase structure); the linguistic tools we employ utilize a subset of the CLAWS tagset [6]. In Fig. 5, the POS-tag of "submits" is VBZ (verb in the "s" form), "item" and "description" are nouns (NN); "Seller" is a proper noun (NNP). Further, as shown in Fig. 5, the *lemma* (base form) of "submits/VBZ" is "submit".

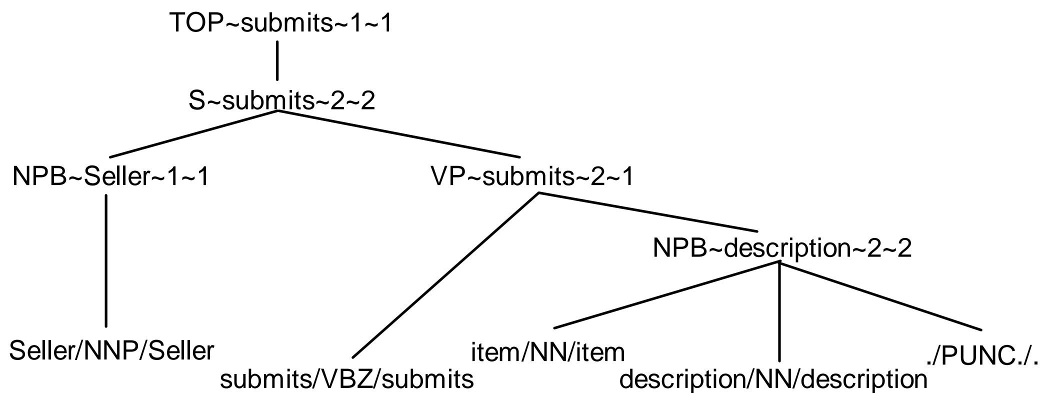

Figure 5: Parse tree of the sentence*: "Seller submits item description"*

Based on the guidelines [7, 11, 18] for writing use cases, we derive the following premises, forming the basis for the conversion:.

Premise 1: An action described by a step of a textual use case describes either (a) communication between an actor and SuD (a request being sent or information passed), or (b) an internal action performed.

Premise 2: Such action is described by a simple English sentence, adhering to a uniform structure pattern.

Support for these premises can be found, e.g., in [7, 11], where it is required that a step describes *"a simple action in which one actor accomplishes a task or passes information to another actor"* or a *"single atomic task"*; sentences should follow the simple structure *"Subject ... verb ... direct object(s) ... preposition ... indirect object(s)"*. Based on these premises, we analyze the parse trees obtained with existing linguistic tools. We propose rules to identify the active entity, the type of action, and the communicating actor; we also identify the principal verb and representative object to select words to construct an event token representing the action.

Subsequently, we construct a finite automaton representing the behavior described by the use case via the main success scenario specification, variations and extensions. Based on this automaton, we create a Pro-case (fig. 6), employing the generic algorithm for transforming a finite automaton into a regular expression.

Of course, as the event tokens obtained are only an estimate of a future method name, and as we are employing a statistical natural language parser, the resulting behavior specification will be inherently imprecise and only is an estimate of the actual behavior specification, but, regardless that, can be of high value to developers.

```
?SL.submitItem ; #validateDescription ;
(  #cond2a
+  ( NULL + #cond2b ; !SL.providePriceAssessment )
   ; ?SL.adjustPrice ; #validateSeller ; #verifySeller ;
   (  #cond5a
   +  !TC.validateOffer ;
      (  #cond6a
      +  #listOffer ; #respondAuthorizationNumber
      )
   )
)
```

Figure 6: Pro-case automatically created for the use case "Seller submits an offer".

We have implemented the conversion of textual use cases into Pro-cases in the *Procasor* tool. The tool employs existing linguistic tools, in particular the statistical natural language parser developed by Michael Collins [8], the Maximum Entropy tagger developed by Adwait Ratnaparkhi [47] and the morphological tool morpha developed by John A. Carroll et al. [35]. For each sentence describing a use case step, the tools yield a parse tree annotated with POS-tags and the lemmas of the words (as shown in Fig. 5. The java-based Procasor tool subsequently processes the parse trees, acquires the principal information describing the action, constructs an event-token for each step, and eventually translates the use case into a Pro-case.

We have applied the Procasor tool to the Marketplace use case model [42] developed within our theoretical work on the Generic UC View formal model, developed without considerations for automatic processing, only following the generic use case writing guidelines. The Pro-cases obtained were very close to Pro-cases originally obtained by hand in [42]; to provide an example, Fig. 6 shows the Pro-case obtained for the use case "Seller submits an offer" (fig. 1), Fig. 4 shows the manually created Pro-case ([43]).

Lessons for use case writers. From the case study we have learned that for textual use cases to be machine-processable, the generally accepted use case writing guidelines have to be adhered to. In particular, the sentences have to be simple, describing only communication between SuD and an actor (or an internal action), and use of synonyms has to be avoided. In addition, it is necessary to avoid relying on context (from previous steps), even where the context would be obvious to a human reader. Thus, use case writers have to learn to write machine-processable textual use cases, but doing so will result into more readable, clear and less ambiguous use cases.

2.5. Analyzing Use Cases in UML 2.0

The emerging standard UML 2.0 features mechanisms to capture relations among use cases as well as to specify behavior of a use case. Therefore, an interesting problem is to which extent these UML mechanisms support behavior assembly, behavior composition, and consistency reasoning.

To analyze the UML 2.0 behavior specification mechanisms, we extended our formal model *Generic UC View*; the main motivation for these extensions is to utilize the features of a behavior specification mechanism where semantics is defined based on traces, while preserving support for other behavior specification mechanisms as well. We address this issue by introducing two specialized extensions of our generic model, *Basic UC View* not concerned with the trace semantics and *Trace-based UC View* particularly tailored for behavior specification mechanisms with trace semantics.

2.5.1. Basic UC View

To facilitate reasoning on different behavior specification mechanism, we identify instances of concepts of the abstract model with a subscript; for a behavior specification mechanism μ, we denote $Scenarios_\mu$ the domain of *μ-behavior* $Com_\mu(A)$ of an entity A; also, $Com_\mu(UC_\mu{}^A{}_i)$ stands for the behavior specified by the use case $UC_\mu{}^A{}_i$ from the domain $U_\mu{}^A$ and $Com_\mu(UC_\mu{}^A{}_i) \subseteq Scenarios_\mu$.

We define *use case expressions* to capture the assembled behavior of an entity A. We assume that use case expression are formed by means of *assembling operators* from the set OP = { + ; | * }, denoting alternative, concatenation, parallel composition, and repetition. To give an operator op $\in$ OP a precise meaning in a specification mechanism μ, we have to associate it with an operation θop: $Scenarios_\mu \times Scenarios_\mu \rightarrow Scenarios_\mu$. This way the behavior described by an expression of the form $Uc_\mu{}^A{}_i$ op $UC_\mu{}^A{}_j$ is $Com_\mu(UC_\mu{}^A{}_i)$ θop $Com_\mu(UC_\mu{}^A{}_j)$. In a similar way, we inductively define the behavior of a general expression of the form $e_{\mu,1}$ op $e_{\mu,2}$ as $Com_\mu(e_{\mu,1}$ op $e_{\mu,2}) = Com_\mu(e_{\mu,1})$ θop $Com_\mu(e_{\mu,2})$. In general, for a particular μ, it can be impossible to associate an operation from $Scenarios_\mu \times Scenarios_\mu \rightarrow Scenarios_\mu$ with each of the operators in OP. Therefore, we introduce $OP_\mu \subseteq OP$ containing all those operators associated with such an operation.

Every behavior specification mechanism μ typically defines native operations upon behavior specifications. In Basic UC View, a native operation f: $U_\mu^A \times U_\mu^A \to U_\mu^A$ combines behavior specifications of use cases into behavior specification of a new use case. To address behavior assembly, we find it very useful to construct a "summary use case" [7] $UC_\mu{}^A{}_i$ from a set of use cases $\{UC_\mu{}^A{}_k\}$ via native operations by using a use case expression e_μ (its syntax tree) as guidelines. Driven by this motivation, we define $UC_\mu{}^A{}_i$ to be a *characteristic use case* of a use case expression e_μ if $Com_\mu(e_\mu) = Com_\mu(UC_\mu{}^A{}_i)$.

In order to construct a characteristic use case $UC_\mu{}^A{}_i$ of e_μ, we have to identify which of the operators from OP_μ have an implementation via native operations of μ. We say that an operation f: $U_\mu^A \times U_\mu^A \to U_\mu^A$ *implements* op $\in OP_\mu$, if for all use cases $UC_\mu{}^A{}_i$, $UC_\mu{}^A{}_j$ it holds that $Com_\mu(UC_\mu{}^A{}_i \text{ op } UC_\mu{}^A{}_j) = Com_\mu(UC_\mu{}^A{}_i \text{ f } UC_\mu{}^A{}_j)$. Here, $UC_\mu{}^A{}_i$ op $UC_\mu{}^A{}_j$ is a use case expression, while $UC_\mu{}^A{}_i$ f $UC_\mu{}^A{}_j$ is a new use case.

If all the operators from OP_μ are implemented by native operations in μ, then, for every use case expression e_μ, there can be constructed its characteristic use case by recursively following its syntactic structure so that $Com_\mu(UC_\mu{}^A{}_k) = Com_\mu(e_\mu)$.

A use case model of an entity A is determined by all its use cases and by the way they are combined (specifying the assembled behavior of A). To capture these concepts formally, we assume that a use case model of A is a pair $<UR_\mu^A, e_\mu^A>$ where $UR_\mu^A \subseteq U_\mu^A$ is the set of relevant use cases ("all its use cases") upon which a use case expression e_μ^A is written. This expression specifies the *assembled behavior* $Com_\mu(e_\mu^A)$ approximating the behavior $Com_\mu(A)$. If there exists a characteristic use case of e_μ^A, we call it *representative use case* of the use case model.

2.5.2. Trace-Based UC View

To capture behavior composition, the content of the domain $Scenarios_\mu$ has to be defined explicitly. To do this, we introduce Trace-Based UC View (an extension of Basic UC View) based on traces. We have chosen this approach because traces are well understood and established in the behavior specification community [4].

In Trace-Based UC View, an activity of A is a finite sequence of atomic events from a finite domain; these events can be represented by event labels from a domain Act_μ (also finite) and a scenario is a *trace* (finite sequence of event labels). Then, $Com_\mu(A) \subseteq Scenarios_\mu = Act_\mu{}^*$.

Assuming an entity C composed of entities A and B (operating in parallel), the goal is to obtain the behavior of C composed of $Com_\mu(A)$ and $Com_\mu(B)$. The event labels of external events of A form the set $Msg_\mu(A) \subseteq Act_\mu$; for entities A and B we define the relation $Pair_\mu^{A,B} \subseteq Msg_\mu(A) \times Msg_\mu(B)$ capturing synchronization of events of A and B. In composed behavior, every pair of synchronized events t^A, t^B is merged into an internal event of C represented by a new event label $\tau t^A t^B$; except for this explicit synchronization, the traces of A and B arbitrarily interleave.

2.5.3. Analyzing UML 2.0: Assembled and Composed Behavior

In UML 2.0, the *UseCase* metaclass interprets the concept of a use case $UC_{\mu}{}^{A}{}_{j}$. A *UseCase* may be associated with a behavior specification of one of the types predefined in UML 2.0: *Activity*, *Interaction*, *ProtocolStateMachine*, and *StateMachine*. In terms of our formal model, these are four distinct concrete use case specification mechanism. In the thesis, for each of them, we provide a brief characteristic and show: (i) how Scenarios are interpreted and whether a scenario is a trace, (ii) how and whether assembled behavior, (iii) representative use case, and (iv) composed behavior can be obtained, and (v) whether consistency reasoning is possible.

We conclude that only *Interactions* define trace-based scenarios and support obtaining assembled behavior, representative, use case as well as composed behavior. On the contrary, behavior of *Activities* and *StateMachines* cannot be interpreted via traces. Assembled behavior can be obtained only for a limited set of assembly operators ($OP_{AV} = OP_{SM} = \{+ ; *\}$). Moreover, while for *StateMachines* the whole OP_{SM} can be implemented via native operations, for *Activities* only "+" is the case. Only for expressions featuring solely these (natively implemented) operators a representative use case can be constructed. Behavior of *ProtocolStateMachines* (PSM) can be captured as traces, however, PSMs are not designed to specify the behavior of an entity, but only of a single *Interface* of a *Port*. Thus, PSMs do not provide an interpretation of composed behavior either.

The conclusion is that none of these mechanisms explicitly addresses assembled behavior, representative use case, nor composed behavior. However, we draw a way to interpret these concepts in *Interactions*.

2.6. Port State Machines

Although Protocol State Machines (PSM) in UML 2.0 [38] describe valid sequences of operation calls, a PSM is applicable only to a single interface, either a provided or required one and cannot capture the call interleaving on multiple interfaces of a Port; moreover, due to the run-to-completion semantics, nested calls cannot be modeled with a PSM. Also, the definition of *protocol conformance* is not precise enough to permit reasoning on the relation in general; thus reasoning on consistency in component composition is not possible with PSMs.

To remedy this problem, we propose the Port State Machine (PoSM) to model the communication on a Port. We base PoSM on UML 2.0 Protocol State Machines [38] and employ the formal model of behavior protocols [44], modeling an operation call as two atomic events *request* and *response*; moreover, the model explicitly distinguishes between *sent* and *received* events. This permits PoSM to capture the interleaving and nesting of operation calls on provided and required interfaces of the Port; further the trace semantics of PoSM yields a regular language. We apply the compliance relation of behavior protocols to PoSMs, allowing us to reason on behavior compliance of components in software architectures; the existing verifier tool can be applied to PoSMs.

Employing the standard UML 2.0 extension mechanisms, PoSM metamodel is defined as an extension to the Protocol State Machine framework. The key metaclasses are *PortStateMachine* (extension of *ProtocolStateMachine*) and *PortTransition* (extension of *ProtocolTransition*); specific constraints are employed to restrict valid PoSMs. A *PortTransition* models either sending or receiving a single event (request or response) for an operation call on an interface; the restrictions guarantee that at most one *PortTransition* may be taken within a single run-to-completion step. Utilizing this property, we define trace semantics of PoSM via *state events* and *communication events* from the respective domains *SE* and *CE*; we capture the behavior specified by a PoSM P^A via its *execution language* $LE(P^A)$ and *communication language* $LC(P^A)$. Conveniently, $LC(P^A)$ is a regular language.

In the behavior model, an operation call handled on a provided interface is represented by a received request event and a sent response event; in a similar way, an operation call issued on a required interface is represented by a sent request event and a received response event. To hide such technical details from the modeler, PoSM notation defines convenient shortcuts: a *call transition* and a *call state*, hiding the necessity to employ intermediate states; these are demonstrated in Fig. 7.

The behavior protocols compliance relation is defined on languages (upon the domain of communication events) and thus, its definition is applicable to PoSMs as well. Although composition and consent are protocol operators, their semantics is defined solely based on the languages generated by their operands and thus, their definition can be extended to communication languages of PoSMs. Conveniently, the language generated by a PoSM is regular (taking into account that there are no constraints, no event deferring and, inherently to state machines, no recursion). Thus, PoSMs permit to establish a compliance relation and apply the behavior protocols compliance verifier [22, 49]. Further, Port State Machines, when considered as an instance of Trace-based UC View, address the consistency issues and can be used to model use cases.

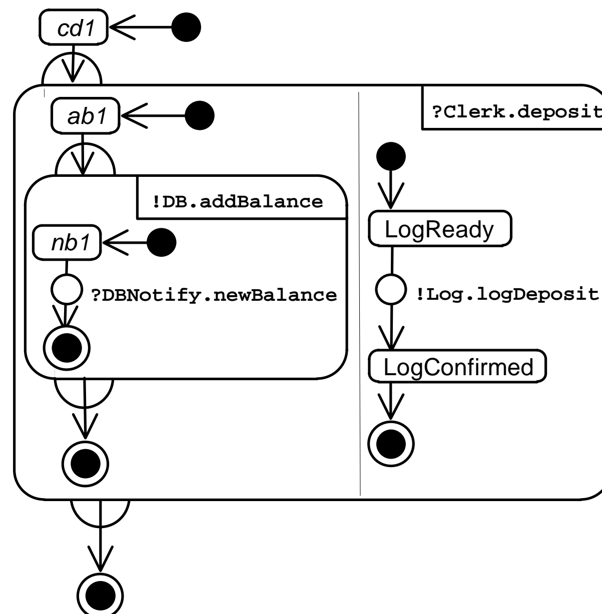

Figure 7: Port State Machine employing *call transition* and *call state* shortcuts

3. Summary of Contribution

3.1. Summary

We have proposed the simple formal model Generic UC View identifying the key abstractions in use case modeling and the relations among them, thus meeting the goals (i) and (ii) outlined in Sect. 1.6. We have applied the model developed to analyze existing approaches, addressing goal (iii) and subsequently, targeting goal (iv), proposed Pro-cases, a new behavior specification mechanisms addressing the issues identified in our analysis. Pro-cases support behavior assembly and composition, as well as consistency reasoning; a tool verifying the compliance relation is available [22,49].

To avoid the duplication of effort in developing Pro-cases and textual use cases in parallel, we have proposed a scheme for converting textual use cases into Pro-cases, following simple guidelines; further, we have employed readily available linguistic tools to automate this conversion. The proposed approach is implemented in the *Procasor* tool, addressing the goal (v); we have evaluated feasibility of our approach by applying the tool to a set of use cases developed within our previous work on the formal model Generic UC View.

With the intension to analyze the emerging standard UML 2.0, we have extended our generic use case model, refining the assembly operations already considered in Generic UC View. In our analysis of UML 2.0, we have explored the options to interpret the assembly operations via native operations of its behavior specification mechanisms, employing their well defined structure and syntax.

The highlights of our findings are that Interactions define trace-based scenarios and support behavior assembly and composition; although semantics of Protocol-StateMachines can also be interpreted with traces, behavior composition is not possible. Behavior of Activities and State Machines cannot be interpreted with traces and their support for behavior assembly is limited. In conclusion, only Interactions satisfy the prerequisites for reasoning on behavior specified in multiple use cases of an entity. This analysis addresses the goal (vi) of the thesis.

Having identified that UML 2.0 StateMachines cannot capture interleaving and nesting of operation calls, we provide a remedy to this problem by proposing Port State Machines, a new behavior specification mechanism based on UML 2.0 Protocol State Machines that fits into UML 2.0 framework. By representing an operation call with two separate events, *request* and *response*, Port State Machines are able to capture the interleaving and nesting of operation calls of provided and required interfaces of a Port, encapsulating the communication of a UML 2.0 component. Conveniently, the behavior compliance relation defined for languages of behavior protocols [44] may be used for languages of PoSM as well; thus, the verifier tool available [22, 49] can be employed to reason on PoSM specifications. By proposing Port State Machines, we have addressed the remaining goal (vii) of this thesis.

3.2. Contribution

The Generic UC View was originally published in the proceedings of the IDPT 2003 conference [41]; at the conference, the paper received the *Rudolf Christian Carl Diesel Best Paper Award*. Subsequently, a slightly modified version of the paper was published in the Transactions of the SDPS: Journal of Integrated Design and Process Science [43]. Further, Generic UC View is also described in TR 02/11 of the Department of Computer Science, University of New Hampshire; the appendices of the technical report feature an elaborate example of use case models.

The recent work on transformation of textual use cases into Pro-cases is still work-in-progress; it has been documented in [26] but has not yet been published.

The same applies to the recent analysis of UML 2.0 behavior specification mechanism, which is still work-in-progress; it has been documented in [34] but has not yet been published.

The Port State Machines, proposed in Chapter 2.6, have been presented at the workshop *Compositional Verification of UML Models*, held as a part of the UML 2003 conference. An extended version of the contribution [28] has been accepted for publication in a special volume of the Electronic Notes in Theoretical Computer Science, published by Elsevier Science. Port State Machines are also described in Tech. Report No. 2003/4 of the Department of Software Engineering, Charles University in Prague [27].

Results from the earlier work on component based software development (not included in the thesis) have been captured in a poster presented at the OOPSLA 2001 conference [24] (extended abstract published in the OOPSLA 2001 Conference Companion) and have been documented in a number of technical reports [30, 29, 31].

References

[1] Adamek, J., Plasil,F.: Behavior Protocols Capturing Errors and Updates. In *Proceedings of the 2nd International Workshop on Unanticipated Software Evolution*, ETAPS, Warsaw, 2003

[2] Allen, R., Garlan, D.: *A Formal Basis for Architectural Connection*, ACM Trans. Softw. Eng. Methodol. 6(3): 213-249 (1997)

[3] Amyot, D., Mussbacher, G.: *On the Extension of UML with Use Case Maps Concepts*, in Proceedings of UML 2000, York, UK, October 2-6, 2000, LNCS 1939, Springer 2000

[4] Bergstra J. A., Ponse A., Smolka S.A.: *Handbook of Process Algebra*, Elsevier 2001, ISBN 0444828303

[5] Bruneton, E. Coupaye, T., Stefani, J.B.: *The Fractal Component Model*, Draft 2.0-3, February 5, 2004, http://fractal.objectweb.org/specification/

[6] CLAWS part-of-speech tagger for English, http://www.comp.lancs.ac.uk/ucrel/claws/

[7] Cockburn, A.: *Writing Effective Use Cases*, Addison-Wesley Pub Co, ISBN: 0201702258, 1st edition, Jan 2000

[8] Collins, M.: *Head-Driven Statistical Models for Natural Language Parsing*, PhD Dissertation, Computer and Information Science Department, University of Pennsylvania, 1999, http://www.ai.mit.edu/ people/mcollins/code.html

[9] Damm, W., Harel, D.: *LSCs: Breathing Life into Message Sequence Charts.* Formal Methods in System Design 19(1): 45-80 (2001), Kluwer 2001

[10] D'Souza, D.: *Components with Catalysis*, http://www.catalysis.org/, 2001

[11] Graham, I.: *Object-Oriented Methods: Principles and Practice*, Addison-Wesley Pub Co, ISBN: 020161913X, 3rd edition December 2000

[12] Grieskamp, W., Lepper, M., Schulte, W., Tillmann, N.: *Testable Use Cases in the Abstract State Machine Language*, in Proceedings of APAQS'01, December 10 – 11, 2001, Hong Kong

[13] Harel, D. *Statecharts: A visual formalism for complex systems*, Science of Computer Programming **8** (1987), pp. 231-274, Elsevier Science Publishers B.V. (North-Holland)

[14] Hausmann, J. H., Hecke, R., Taentzer, G.: *Detection of Conflicting Functional Requirements in a Use Case-Driven Approach*, ICSE 2002, Orlando, FL, USA, May 19-25, 2002

[15] Hurlbut R. R.: *A Survey of Approaches For Describing and Formalizing Use Cases*, Expertech, Ltd., Document: XPT-TR-97-03

[16] Jacobson, I., Christerson, M.: *A Growing Consensus on Use Cases*, JOOP 8(1): 15-19 (1995)

[17] Jacobson, I., *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Pub Co; ISBN: 0201544350; 1st edition (June 30, 1992)

[18] Kulak, D., Guiney, E.: *Use cases: requirements in context*, Addison-Wesley, Pub Co, ISBN: 0-201-65767-8, May 2000

[19] Larman, C.: *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall PTR, ISBN: 0130925691, 2nd ed, 2001

[20] Latella, D., Majzik, I., Massink, M.: *Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the SPIN Model-checker*, Formal Aspects of Computing vol 11 no **6** (1999), pp. 637-664

[21] Leavens, G.T., Sitaraman, M. (eds.): *Foundations of Component-Based Systems*, Cambridge University Press, March 2000, ISBN: 0521771641

[22] Mach, M., Plasil, F.: *Addressing State Explosion in Behavior Protocol Verification*, Accepted for publication in proceedings of SNPD'04, Beijing, China, Jun 2004

[23] Magee, J., Kramer, J.: *Dynamic Structure in Software Architectures*, in Proceedings of SIGSOFT FSE 1996, San Francisco, California, USA, October 16-18, 1996. ACM SIGSOFT Software Engineering Notes 21(6), November 1996

[24] Mencl, V.: *Autonomous Points in Component Composition*, Extended abstract of the Poster presented at OOPSLA 2001, in the Conference Companion, ACM ISBN: 1-58113-441-X, Tampa, FL, USA, Oct 2001

[25] Mencl, V.: Component Definition Language, Master Thesis, advisor: Nguyen Duy Hoa, Dept. of SW Engineering, Faculty of Mathematics and Physics, Charles University in Prague, 1998

[26] Mencl, V.: *Converting Textual Use Cases into Behavior Specifications*, work-in-progress

[27] Mencl, V.: *Enhancing Component Behavior Specifications with Port State Machines*, Tech. Report No. 2003/4, Dep. of SW Engineering, Charles University, Prague, Sep 2003

[28] Mencl, V.: *Specifying Component Behavior with Port State Machines*, Accepted for publication in proceedings of Compositional Verification of UML Models workshop (Oct 21, 2003, part of UML 2003) in a volume of the Electronic Notes in Theoretical Computer Science, Elsevier Science

[29] Mencl, V., Adamek, J., Buble, A., Hnetynka, P., Visnovsky, S.: *Enhancing EJB Component Model*, Tech. Report No. 2001/7, Dep. of SW Engineering, Charles University, Prague, Dec 2001

[30] Mencl, V.: *Managing Configuration of Update-enabled Software Components*, Tech. Report No. 2001/5, Dep. of SW Engineering, Charles University, Prague, Oct 2001

[31] Mencl, V., Hnetynka, P.: *Managing Evolution of Component Specifications using a Federation of Repositories*, Tech. Report No. 2001/2, Dep. of SW Engineering, Charles University, Prague, Jun 2001

[32] Mencl, V., Hnetynka, P.: Managing Type Information in an Evolving Environment, Week of Doctoral Students WDS 2000, Faculty of Mathematics and Physics, Jun 2000

[33] Mencl, V., Petrova, Z., Plasil, F.: Update description language (position paper), Week of Doctoral Students WDS 99, Faculty of Mathematics and Physics, Jun 1999

[34] Mencl, V., Plasil, F., Adamek, J.: *Use Cases in UML 2.0: Analyzing Support for Behavior Assembly and Composition*, work-in-progress

[35] Minnen, G., Carroll J., Pearce, D.: *Applied morphological processing of English*, Natural Language Engineering, 7(3), pp. 207-223, (2001), http://www.cogs.susx.ac.uk/lab/nlp/carroll/morph.html

[36] Object Management Group (OMG): Unified Modeling Language (UML), version 1.4, formal/2001-09-67, http://www.omg.org/uml/

[37] Object Management Group (OMG): Unified Modeling Language (UML), version 1.5, formal/2003-03-01, http://www.omg.org/uml/

[38] Object Management Group (OMG): Unified Modeling Language: Superstructure, version 2.0, Final Adopted specification, ptc/03-08-02, http://www.omg.org/uml/

[39] Odeh, M., Hauer, T., McClatchey, R., Solomonides, T.: *A Use-Case Driven Approach in Requirements Engineering : The Mammogrid Project*, Presented at the 7th IASTED Int Conf on Software Engineering Applications, Marina del Rey, USA November 2003, arXiv:cs.DB/0402008, http://arxiv.org/abs/cs.DB/0402008, Feb 2004

[40] Plasil, F., Balek, D., Janecek, R.: *SOFA/DCUP Architecture for Component Trading and Dynamic Updating*, In *Proceedings of ICCDS '98*, Annapolis, IEEE Computer Soc. (1998)

[41] Plasil, F., Mencl, V.: *Getting "Whole Picture" Behavior in a Use Case Model*, in Proceedings of IDPT 2003, Austin, Texas, U.S.A., Dec 2003, ISSN 1090-9389

[42] Plasil, F., Mencl, V.: *Use Cases: Assembling "Whole Picture Behavior"*, TR 02/11, Dept. of Computer Science, University of New Hampshire, Durham, 2002

[43] Plasil, F., Mencl, V.: *Getting "Whole Picture" Behavior in a Use Case Model*, in Transactions of the SDPS: Journal of Integrated Design and Process Science, vol. 7, no. 4, pp. 63-79, Dec 2003, ISSN-1092-0617

[44] Plasil F., Visnovsky, S.: *Behavior Protocols for Software Components*. Transactions on Software Engineering, IEEE, vol 28, no **11** (2002)

[45] Plasil, F., Visnovsky, S., Besta, M.: *Bounding Behavior via Protocols*, Proceedings of TOOLS USA '99, Santa Barbara, CA, Aug. 1999.

[46] Pospisil, R., Prochazka, M., Mencl, V.: On Performance of Enterprise JavaBeans, Presented at the Objekty'99 conference, Prague, Nov 1999

[47] Ratnaparkhi, A.: *A Maximum Entropy Part-Of-Speech Tagger*, in Proceedings of the Empirical Methods in Natural Language Processing Conference, May 17-18, 1996. University of Pennsylvania, http://www.cis.upenn.edu/~adwait/statnlp.html

[48] Schneider, G., Winters, J. P.: *Applying Use Cases: A Practical Guide*, Addison-Wesley Pub Co, ISBN: 0201708531, 2nd edition, March 2001

[49] SOFA Behavior Protocol Verifier, SOFA project, http://nenya.ms.mff.cuni.cz/projects/sofa/tools/

[50] Stevens, P.: *On Use Cases and Their Relationships in the Unified Modelling Language*, in Proceedings of FASE 2001 (part of ETAPS 2001), Genova, Italy April 2001, Springer LNCS 2029

Author’s Publications

Reviewed articles

[41] Plasil, F., Mencl, V.: *Getting “Whole Picture” Behavior in a Use Case Model*, in Proceedings of IDPT 2003, Austin, Texas, U.S.A., Dec 2003, ISSN 1090-9389, received the *Rudolf Christian Carl Diesel Best Paper Award.*

[43] Plasil, F., Mencl, V.: *Getting "Whole Picture" Behavior in a Use Case Model*, in Transactions of the SDPS: Journal of Integrated Design and Process Science, vol. 7, no. 4, pp. 63-79, Dec 2003, ISSN-1092-0617

[28] Mencl, V.: *Specifying Component Behavior with Port State Machines*, Accepted for publication in proceedings of Compositional Verification of UML Models workshop (Oct 21, 2003, part of UML 2003) in a volume of the Electronic Notes in Theoretical Computer Science, Elsevier Science

[24] Mencl, V.: *Autonomous Points in Component Composition*, Extended abstract of the Poster presented at OOPSLA 2001, in the Conference Companion, ACM ISBN: 1-58113-441-X, Tampa, FL, USA, Oct 2001

Nonrefereed publications

[32] Mencl, V., Hnetynka, P.: *Managing Type Information in an Evolving Environment*, Week of Doctoral Students WDS 2000, Faculty of Mathematics and Physics, Jun 2000

[46] Pospisil, R., Prochazka, M., Mencl, V.: *On Performance of Enterprise JavaBeans*, Presented at the Objekty'99 conference, Prague, Nov 1999

[33] Mencl, V., Petrova, Z., Plasil, F.: *Update description language (position paper)*, Week of Doctoral Students WDS 99, Faculty of Mathematics and Physics, Jun 1999

Master thesis

[25] Mencl, V.: *Component Definition Language*, Master Thesis, advisor: Nguyen Duy Hoa, Dept. of SW Engineering, Faculty of Mathematics and Physics, Charles University in Prague, 1998

Work in progress to be submitted

[34] Mencl, V., Plasil, F., Adamek, J.: *Use Cases in UML 2.0: Analyzing Support for Behavior Assembly and Composition*, work-in-progress

[26] Mencl, V.: *Converting Textual Use Cases into Behavior Specifications*, work-in-progress

Technical reports

[27] Mencl, V.: *Enhancing Component Behavior Specifications with Port State Machines*, Tech. Report No. 2003/4, Dep. of SW Engineering, Charles University, Prague, Sep 2003

[42] Plasil, F., Mencl, V.: *Use Cases: Assembling “Whole Picture Behavior”*, TR 02/11, Dept. of Computer Science, University of New Hampshire, Durham, Nov 2002

[29] Mencl, V., Adamek, J., Buble, A., Hnetynka, P., Visnovsky, S.: *Enhancing EJB Component Model*, Tech. Report No. 2001/7, Dep. of SW Engineering, Charles University, Prague, Dec 2001

[30] Mencl, V.: *Managing Configuration of Update-enabled Software Components*, Tech. Report No. 2001/5, Dep. of SW Engineering, Charles University, Prague, Oct 2001

[31] Mencl, V., Hnetynka, P.: *Managing Evolution of Component Specifications using a Federation of Repositories*, Tech. Report No. 2001/2, Dep. of SW Engineering, Charles University, Prague, Jun 2001

Presentations

Mencl, V.: *Specifying Component Behavior with Port State Machines*, presented at Colloquium CIS-TU Berlin and Fraunhofer-ISST, Berlin, Apr 2004

Mencl, V.: *From Textual Use Cases to Behavior Specifications*, presented at Colloquium CIS-TU Berlin and Fraunhofer-ISST, Berlin, Apr 2004

Plasil, F., Mencl, V.: *Getting "Whole Picture" Behavior in a Use Case Model*, presented at CS900 Colloquium seminar, Dept. of CS, University of New Hampshire, Durham, NH, U.S.A., Dec 2003

Plasil, F., Mencl, V.: *Getting "Whole Picture" Behavior from Use Cases*, presented by F. Plasil at Informatics Colloquium, Masaryk University, Brno, Apr 2003

Citations

[25] is cited in:

- Crevola, F.: Décrire une application à base de composants sur middleware, Université de Nice Sophia - Antipolis, Encadrement: Michel Riveill, June 2001, http://rainbow.essi.fr/publications/01-dea-crevola.pdf
- Rovner, J., Valdman, J.: SOFA Review : experiences from implementation, in Proceedings of ISM'01, Information Systems Modelling, Ostrava : MARQ, 2001. - ISBN 80-85988-51-8
- Rovner, J.: *Fault tolerant SOFA framework proposal*, in Proceedings of Objekty '2002, Praha, 2002, ISBN 80-213-0947-4

[33] is cited in: Beniya, J., Kobayashi, Y., Myong, K. H., Nakayama, K., Maekawa, M.: Data structure adaptation for dynamic software updating, Graduate School of Information Systems, University of ElectroCommunications, Proceedings of the 65th Annual Conference of Information Processing Society of Japan (IPSJ), 2003, (to appear), http://www.maekawa.is.uec.ac.jp/os/aya/paper/beniya-ipsj2003.pdf

[24] is cited in:

- Kniesel, G.: Seminar "Component Engineering, Wintersemester 2003", http://www.inf.uos.de/kniesel/lehre/ws03/compEng/literatur.html
- Cremers, A. B., Kniesel, G., Cifka, M.: Seminar: "Component Engineering for Media Applications", http://www.informatik.uni-bonn.de/III/lehre/seminare/SWT/SS2003/
- Peking University SSCC research seminar, http://sscc.pku.edu.cn/docsearch.asp?showsubject=PK,12,2