*Electronic Notes in Theoretical Computer Science 50 No. 3 (2001) { Proc. GT-VMT 2001 URL:* [*http://www.elsevier.nl/locate/entcs/volume50.html*](http://www.elsevier.nl/locate/entcs/volume50.html) *9 pages*

*Behavioral Constraints for Visual Models*

*Reiko Heckel 1 Jochen M. Kuster 2 Dept. of Mathematics and Computer Science*

*University of Paderborn*

*D-33095 Paderborn, Germany*

*Abstract*

*In this paper, we discuss the issue of consistency of behavioral models in the UML and present techniques for specifying and analyzing consistency. Using meta-model rules we transform elements of UML models into a semantic domain. Then, consis- tency constraints can by speci ed and validated using the language and the tools of the semantic domain. This general methodology is exempli ed by the problem of protocol statechart inheritance.*

*Key words: meta modeling, model veri cation, behavioral consistency*

# *1 Introduction*

*As a general-purpose modeling language, the UML [11] lacks precise guidelines* of how to use certain diagrams in the development process. Instead, mecha- nisms are provided to de ne domain or project-speci c specializations and dialects (called pro les) and each dialect may come with its own methodol- ogy. As a consequence, the semantic overlap between di erent diagrams or submodels cannot be xed once and for all, but depends on the dialect in question. To some extend, the meta modeling approach [10] used to de ne the abstract syntax and static semantics of the UML can be used for specifying the additional syntactic elements and the structural consistency constraints associated with a UML dialect. However, so far there exists no general (i.e., meta level) techniques for specifying the behavioral consistency for the UML and its dialects.

*Research partially supported by the ESPRIT Working Group APPLIGRAPH and the*

*TMR network GETGRATS.*

*1 Email:* [*reiko@upb.de*](mailto:reiko@upb.de)

*2 Email:* [*jkuester@upb.de*](mailto:jkuester@upb.de)

*c 2001 Published by Elsevier Science B. V.* Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

down



prog1

prog2

up

down

up

up

down

up

prog4

prog3

down

|  |
| --- |
| Tv |
|  |
| up  down time |

time

|  |
| --- |
| TvRemote |
|  |
| ch1  ch2 ch3 ch4 |



ch1

ch2

down

prog1

prog2

up

down

up

up

down

time

up

prog4

prog3

down

ch4

ch3

*Fig. 1. Invocable behavior: The Tv example*

*It is the aim of the paper to outline an approach to the speci cation and* veri cation of behavioral constraints for visual models. We proceed in four steps. After identifying (informally) the consistency problem at hand (Sect. 2), we choose a semantic domain which supports the kind of consistency prob- lem we are interested in and de ne a mapping of models into the semantic domain (Sect. 3). Then, we use the language and tools provided by the se- mantic domain to formulate the behavioral constraints and to verify them

*w.r.t. individual models (Sect. 4). In the rest of this paper, we exemplify our*

*approach by formulating two notions of UML statechart inheritance through* a rule-based mapping into CSP.

# *2 UML Statecharts and Inheritance*

*In the UML, a statechart can be associated to a class in order to specify* the object life cycle, i.e., the order of operations called upon an object of this class during its life-time. Given a class A and a subclass B of A, the behavioral conformity of the associated statecharts gives rise to the problem of statechart inheritance. In the literature, di erent notions are proposed (see, e.g., [3,2,7]). In this paper, we will restrict ourselves to two notions related to two dual interpretations of statecharts as specifying invocable or observable behavior. Invocable consistency. This notion of statechart inheritance is based on the substitution principle requiring that an object of class B can be used where an object of class A is required. This means, any sequence of operations invocable on the superclass can also be invoked on the subclass. As an example, consider the situation depicted in Figure 1. Here, any sequence of operations invocable on a Tv object can also be invoked on a TvRemote object, as the statechart of the former is completely included in the statechart of the latter [3].

*Observable consistency. A dual notion of statechart inheritance is based on the*

getsMarried

|  |
| --- |
| Person |
|  |
| birthday getsMarried getsEngaged |



single

married

getsDivorced

birthday

getsDivorced

TraditionalPerson

birthday



single

getsMarried

getsEngaged

notengaged

engaged

married

birthday

birthday

getsDivorced

birthday

*Fig. 2. Observable behavior: The Person example*

*idea that a statechart is interpreted as description of (an upper bound to) the* observable sequences of method calls. Hence, each sequence observable with respect to a subclass must result (under projection to the methods known) in an observable sequence of its superclass. As an example, consider Figure 2 where, by ltering out the getsEngaged method, the behavior of TraditionalPer- son objects can be projected onto that of Person objects [3].

# *3 Abstract Syntax and Denotational Semantics*

*Meta model. The abstract syntax of the UML diagrams we are using as an* example is speci ed by the meta model in Figure 3. It de nes a simpli ed no- tion of statecharts and their association with classes. Moreover, Generalization (between classes) is modeled. The presentation conforms to the UML meta model but for the attening of some inheritance relations and the introduc- tion of one derived attribute events which shall contain the set of all events belonging to the internal transitions of a CompositeState. All meta classes con- tain a meta attribute name:string which is not shown in the gure. (In the UML meta model this is inherited from the super class ModelElement).

*CSP as semantic domain. Communicating Sequential Processes (CSP) [8] pro-* vide a mathematical model for concurrency based on a simple programming notation and supported by tools [5]. In fact, the existence of language and tool support are most important to our aim of specifying and verifying consistency constraints, despite the existence of more expressive mathematical models. Next, we brie y review the syntax and semantics of the CSP processes we are using.

*Given a set A of actions and a set of process names N , the syntax of CSP*

source

0..1 trigger

*Fig. 3. UML meta model fragment: protocol statecharts and generalization is given by*

Transition

1

StateVertex

Event

subvertex

target

internal

Generalization

parent 1 1 child

Class

context

PseudoState

State top 0..1 StateMachine

1

0..1

isConcurrent: Bool events: Set(string)

FinalState

SimpleState

CompositeState

kind: PStateKind

*P ::= stop j a ! P j P u P j P 2 P j P n a j pn*

*where a 2 A, A A, and pn 2 N . Process names are used for de ning recur-* sive processes using equations pn = P . The interpretation of the operations is as follows. stop represents the inactive (deadlocked) process. The pre x pro- cesses a ! P performs action a and continues like P . The processes P uQ and P 2 Q represent internal and external choice between P and Q, respectively. That means, while P u Q performs an internal ( -)action when evolving into P or into Q, for P 2 Q this requires an observable action of either P or Q. For example, (a ! P ) u (b ! Q) performs in order to become either a ! P or b ! Q. Instead, (a ! P ) 2 (b ! Q) must perform a or b and evolves into P or Q, respectively. This distinction shall be relevant for the translation of statechart diagrams below. Finally, the process P n a behaves like P except that all actions a are hidden.

*The semantics of CSP is usually de ned in terms of traces and failures. A trace is just a nite sequence s 2 A of actions which may be observed when* a process is executing. A failure (s; A) provides, in addition, a set A A of actions that can be refused by the process after executing s. The traces of a process are always closed under pre xes. Therefore, they can only capture safety properties of processes, i.e., properties that are also valid for the inactive process stop. In addition, the failures model can capture lifeness conditions like freedom from deadlocks.

*Together with the two semantic models come two notions of process re-*

*nement. We write P vT Q if T (Q) T (P ), i.e., every trace of Q is also* a trace of P . Analogously, P vF Q if the failures of Q are included in the failures of P . In general, the idea is that P is a re nement of Q if P is more deterministic (less speci ed) than Q. These re nement relations shall be used to express consistency requirements.

*Mapping statecharts to CSP. The translation of statecharts into CSP processes*

State(fin) = stop

**(1)**

**(5)**

beh(s) ::=

**(2)**

name = comp

subvertex

directBeh(s)[] beh(comp)

**(3)**

source

trigger

**(6)**

directBeh(s) ::= event(e1)

[] ...

[] event(en)

**(7)**

name = s

source

event(e) ::=

e −> State(s1)

|~| ...

|~| State(sn)

trigger

**4)**

target

name =

{s1, ..., sn}

:State

name = e

:Transition

:Event

:State

name =

{e1, ..., en}

:Transition

:Event

name = s

:State

name = s

:StateVertex

:CompositeState

name = fin

:FinalState

name = s

:SimpleState

name = comp isConcurrent = false events = E

:CompositeState

State(s) = beh(s)

State(comp) = State(default)

name = default

:State

target

name = init kind = initial

:PseudoState

source

:Transition

beh(s) ::= **(**

:StateMachine

top

:State name = s

stop

*Fig. 4. Mapping rules for state decomposition (1-3) and behavior (4-7)*

*is described by the rules in Figure 4 based on the meta model in Figure 3. The* strategy is as follows. First, the rules (1) to (3) create a system of recursive equations, one for every instance of meta class State. Next, the rules (4) to

*(7) are used to replace all occurrence of the auxiliary process name beh(s)*

*(introduced in rule (2)) by corresponding process de nitions. In general, all* names set in italics represent \non-terminals" that have to be replaced. Notice that we have used the machine-readable version of the CSP notation where [] and |~| denote 2 and u, respectively.

*Observe that the rules (6) and (7) contain multi-objects (denoted by the* shaded borders) which represent maximal sets of concrete objects. As a conse- quence, their attributes deliver sets of values, in our case the sets of names of all events fe1; : : : ; eng or states fs1; : : : ; sng meeting the structural require- ments.

*Below, the application of these rules to the statechart of class Tradition-* alPerson is shown. (The name of the class is abbreviated to TP.) Notice that (\*) beh(single) = directBeh(single)[]beh(top) = beh(top) = STOP by rule (5,4,6) and the CSP axiom p 2 STOP = p. That means, the external behavior of the implicit top state (which is not visible in the concrete syntax and does not have outgoing transitions or super-states) is empty, and the same holds for state simple which does not have outgoing transitions either. Therefore, the semantics of state single is de ned by rule (3) to be that of the default state notengaged. As notengaged is a SimpleState, rules (2) and (5) are applied. Af- ter dropping the super-state component using (\*), we just collect the outgoing transitions using rules (6) and (7). The semantics of engaged and married is computed in a similar way.

top

name =top1 events = E1

:StateMachine

:State

context

parent

name = C1

:Class

child

context

name = C2

:Class

Generalization

top

name =top2 events = E2

:StateMachine

:State

*Fig. 5. Constraints for generalization. Invocable consistency: C2(top2 ) vT C1 (top1 ). Observable consistency: C1(top1 ) vT C2(top2 ) n (E2 E1).*

*TP(single) = TP(notengaged ) (3)*

*TP(notengaged ) = beh(notengaged ) (2)*

*= directBeh(notengaged )[]beh(single) (5)*

*= directBeh(notengaged ) ( )*

*= event(birthday)[]event(getsEngaged ) (6)*

*= birthday ! TP(notengaged )[]getsEngaged*

*! TP(engaged ) (7)*

*TP(engaged ) = beh(engaged ) (2)*

*= directBeh(engaged ) (5; )*

*= getsMarried ! TP(married )[]birthday*

*! TP(engaged ) (6; 7)*

*TP(married ) = directBeh(married ) (2; 5; )*

*= getsDivorced ! TP(notengaged )[]birthday*

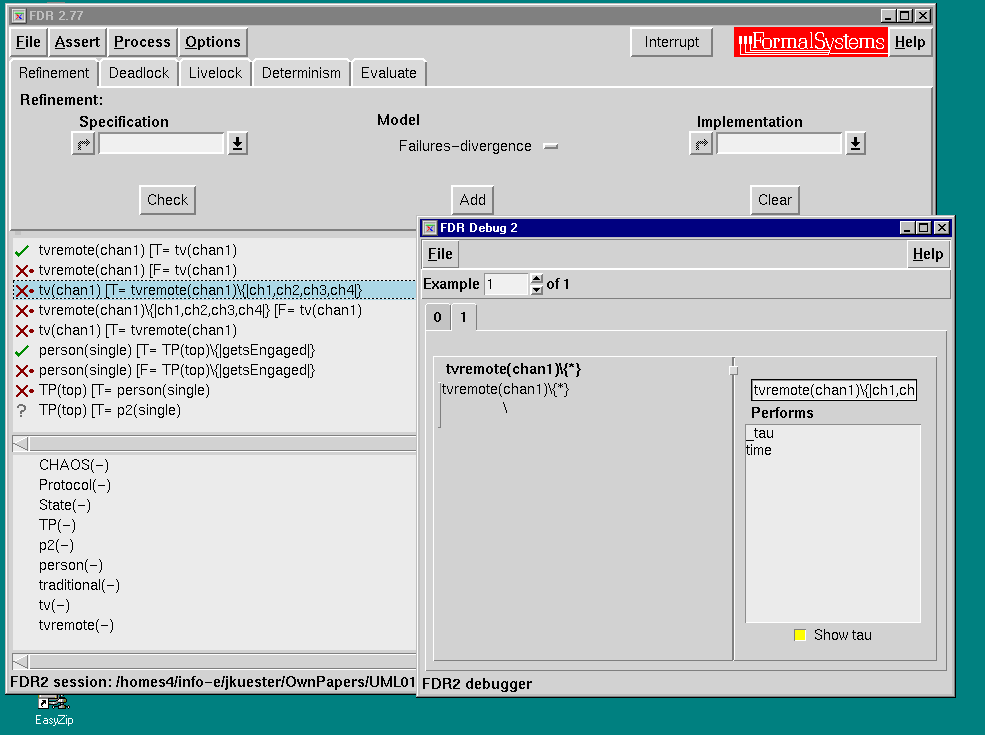
*! TP(married ) (6; 7)*

# *4 Behavioral Constraints*

*Speci cation. Based on the mapping of statecharts into CSP, formal consis-* tency conditions for the two notions of statechart inheritance discussed in Section 2 can be formulated. Both use the pattern of Figure 5. In order to capture the idea of invocable consistency, that each sequence of method calls accepted by the superclass should also be implemented by the subclass, we require that the former is a re nement of the latter. (Recall that P vT Q i T (Q) T (P )).

*If the statechart of the superclass represents an upper bound to the ob-* servable behavior of the subclasses, we have to specify a dual condition hiding with n(E2 E1) all operations newly introduced in the subclass.

*Veri cation. Using the FDR tool, such consistency constraints can be an-*



*Fig. 6. A screenshot of the FDR tool showing a counterexample*

*alyzed. Concerning observable behavior, we can show that P (single) vT* TP(single) n getsEngaged . This means that all traces of the traditional person are included in the set of traces of the person where operations in traces of the traditional person not de ned for the person (in our case the getsEngaged oper- ation) are hidden. With respect to invocable sequences of methods, TP(single) is not a consistent re nement of P(single) because the trace getsMarried is not invocable on instances of TraditionalPerson.

*For the Tv example, we can show that TvRemote(prog 1) vT Tv (prog 1).* This implies that any sequence of operations invoked on a Tv object is also an invocable sequence for a TvRemote object. On the contrary, TvRemote is not observably consistent with Tv because Tv (prog 1) vT TvRemote(prog 1) cannot be established. This is because the restriction of the trace up ch4 time up invocable on TvRemote yields the trace up time up which is not a trace of Tv.

*Figure 6 shows a screenshot of the FDR tool showing this example. The* tool is essentially a model checker verifying re nements between two CSP expressions. If this relation does not hold, a trace or failure is produced as a counterexample. It is evident that, in order to make our approach usable, an interface will be required which presents such counterexamples in UML notation.

# *5 Conclusion*

*In this paper, we have proposed a methodology for specifying and analyzing* behavioral constraints in visual modeling techniques, based on a mapping of models into a semantic domain with language and tool support. This approach is not restricted to behavioral consistency constraints. For example, [6] ana- lyze consistency of cardinality constraints in structural diagrams based on a translation into a system of linear inequalities. Another approach, also based on solving systems of linear inequalities, is [9] who analyze timing constraints of sequence diagrams.

*Rule-based mappings, like the one in Section 3, are also used in [12,1] where* timed Petri nets are proposed as a semantic framework for the UML. Notice, however, that it is not our aim to provide a denotational semantics for the UML (or even a reasonable sublanguage of it). On the contrary, the mapping is de ned locally for the language features of interest, even if the semantics of other model elements is not yet clari ed.

*In order to be able to modify the notion of consistency (when the de-* velopment process evolves or a new pro le is created), it is important that this mapping is exible and extensible. We think that the rule-based nota- tion, which was already used in [4] for describing Java code generation and is originally motivated by pair grammars [13], provides a good starting point. However, it has to be supported by a tool which is able to generate a translator from such a rule-based description. Currently, we are investigating the use of XSL transformations for this purpose.

# *References*

*[1] Baresi, L. and M. Pezz e, Improving UML with Petri nets, in: Proc. ETAPS2001 Workshop on Uniform Approaches to Graphical Process Speci cation Techniques (UniGra), Genova, Italy, Electronic Notes in TCS (2001), to appear.*

*[2] Ebert, J. and G. Engels, Structural and behavioral views of OMT-classes, in:*

*E. Bertino and S. Urban, editors, Proceedings, Object-Oriented Methodologies and Systems, LNCS 858 (1994), pp. 142{157.*

*[3] Ebert, J. and G. Engels, Speci cation of Object Life Cycle De nitions, Fachberichte Informatik 19{95, Universitat Koblenz-Landau (1995).*

*URL http:*

[*//www.uni-koblenz.de/fb4/publikationen/gelbereihe/RR-19-95.ps.gz*](http://www.uni-koblenz.de/fb4/publikationen/gelbereihe/RR-19-95.ps.gz)

*[4] Engels, G., R. Hucking, S. Sauer and A. Wagner, UML collaboration diagrams and their transformation to Java, in: R. France and B. Rumpe, editors, Proc. UML'99, Fort Collins, CO, USA, LNCS 1723 (1999), pp. 473{488.*

*[5] Formal Systems Europe (Ltd), \Failures-Divergence-Re nement: FDR2 User Manual," (1997).*

*[6] Fradet, P., D. L. M etayer and M. P erin, Consistency checking for multiple view software architectures, in: O. Nierstrasz and M. Lemoine, editors, ESEC/FSE '99, Lecture Notes in Computer Science 1687 (1999), pp. 410{428.*

*[7] Harel, D. and O. Kupferman, On the Inheritance of State-Based Object Behavior, Technical Report MCS99-12, Weizmann Institute of Science, Faculty of Mathematics and Computer Science (1999).*

*[8] Hoare, C. A. R., \Communcating Sequential Processes," Prentice Hall, 1985.*

*[9] Li, X. and J. Lilius, Timing analysis of UML sequence diagrams, in: R. France and B. Rumpe, editors, UML'99 - The Uni ed Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings, LNCS 1723 (1999), pp. 661{674.*

*[10] Object Management Group, Meta object facility (MOF) speci cation (1999),* [*http://www.omg.org.*](http://www.omg.org/)

*[11] Object Management Group, UML speci cation version 1.3 (1999), http://* [*www.omg.org.*](http://www.omg.org/)

*[12] Pezz e, M. and L. Baresi, Can graph grammars make formal methods more human?, in: A. Corradini and R. Heckel, editors, Proc. ICALP2000 Workshop on Graph Transformation and Visual Modelling Techniques, Geneva, Switzerland (2000),* [*http://www.di.unipi.it/GT-VMT/.*](http://www.di.unipi.it/GT-VMT/)

*[13] Pratt, T. W., Pair grammars, graph languages and string-to-graph translations, Journal of Computer and System Sciences 5 (1971), pp. 560{595.*