On Combining Semi-Formal and Formal Object Specification Techniques

Martin Gogolla, Mark Richters
University of Bremen, Informatics Department
Postfach 330440, D-28334 Bremen, Germany

Abstract. In the early phases of software development it seems profitable to freely mix semi-formal and formal design techniques. Formal techniques have their strength in their ability to rigorously define desired software qualities like functionality, whereas semi-formal methods are usually said to be easier to understand and to be more human-nature oriented. We propose a new approach in order to combine these two areas by exploiting how constructs of the formal specification language TROLL light are related to the graphical elements of the UML approach.

1 Introduction

As a first step and in order to explain the roles of and the relationships between semi-formal and formal specifications, let us start with a very simple software process model. We assume software development starts somehow with stating the requirements of the software system. These requirements are then formalized in the specification phase and are made more precise with respect to implementation in the design phase. In the implementation phase the design is translated to concrete code, and in the maintenance phase the software is adjusted to evolving user needs. In all phases backtracking to earlier phases is possible.

Now consider the different description techniques used in these phases in reverse order. Maintenance and implementation mainly have to handle programming language code. Design and specification deal with specification language texts. Requirements are usually formulated in natural language thus utilizing informal description techniques. The main link between informal and formal descriptions are often so-called semi-formal techniques. Frequently, diagrams are employed to visualize certain system aspects. The transition from informal to formal descriptions is one of the most crucial points in software development because all later phases rely on the first formal specification. We concentrate on this topic in translating a formal specification language into the Unified Modeling Language UML [BJR97]. Thus we take advantage of the visual power (and the informal semantics) of UML in order to represent aspects of TROLL light specifications having equivalent formal semantics [GH95] as the corresponding UML constructs.

Our approach is very similar to [LR97]. The intention there however seems to be to have a new alternative, complete, and graphical specification formalism whereas we only want to visualize certain aspects with a well-established notation. Earlier work related to ours is [JWH+94]. That approach however considers one of the UML predecessors OMT [RBP+91]. It does not treat for instance activity diagrams, and handles object interaction aspects in a different way. The object specification language GNOME [RS95] also had a graphical representation but it did not draw strong connections to established techniques. Close to our intention is also work [BHH+97] in the direction of giving a formal semantics for UML.

The structure of the rest of this paper is as follows. In Sect. 2 we explain our view on the general relationship between semi-formal and formal specifications. Section 3 introduces a special case of this, namely the way we combine TROLL light specifications with the UML software development approach. The paper ends with some concluding remarks.

2 Relating semi-formal and formal specifications

A closer look to the early software development phases where semi-formal and formal descriptions are mixed is displayed in Fig. 1. We do not assume that the development steps mentioned before are carried out in the sketched order but that requirement and specification steps are freely interchanged. In our approach semi-formal descriptions of software systems are built with the central aim of achieving a formal specification. They concentrate on certain particular aspects of the system to be described. An aspect is obtained by considering only very special properties of the system to be developed (an aspect in this sense is not be understood as a component or module of the system). For example, one aspect in this sense may be the structural system aspect (what objects are in the system and how are they related) and another one may be the dynamic system aspect (how do these objects evolve in time). These aspects correspond to what is known in the database field as a view on the underlying basic unit. There may be many such aspects or views, e.g. an alpha, beta, ..., omega view. These views in turn can be subject to focusing on certain properties only (views can be built on views and so on). For example, in Fig. 1 we have views alpha-1, ..., alpha-n which rely on the alpha view.

In the software development process, of course, one does a lot of transformation respectively modification of semi-formal and formal descriptions, and it is challenging to ask to what extent the previously achieved results can be used further. In the first line one has to relate formal descriptions and semi-formal ones. Translations in both directions can make sense. In going from formal to semi-formal descriptions you give for instance later system users the chance to understand the planned system functionality or you allow for designers to communicate on a short cut basis. In going from semi-formal to formal specifications you make precise what has been achieved for instance in discussions with later

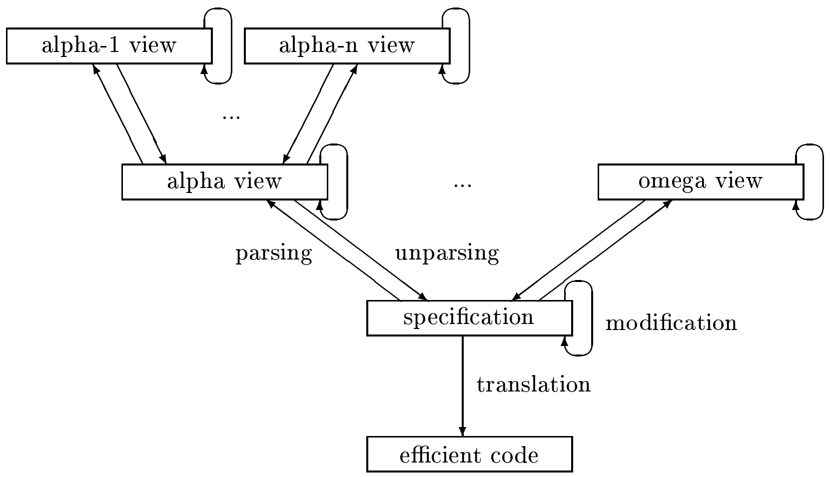

Figure 1. Semi-formal views on a specification

system users. Most interesting is a situation where you already have a certain development state with consistent formal and semi-formal descriptions and you make modifications respectively further developments on the semi-formal level. The challenging question then of course is to consistently incorporate these newly added design decisions into the already well-developed formal specification.

So, the overall aim of our approach is to give support for the development of formal specifications by means of graphical description techniques. But on the other hand we do not insist on pure graphical terms in the sense that we expect that everything must be expressed in a graphical notion, i.e. we do not expect that the views cover the complete specification. There will be certain parts that are easier to express in textual form, for example complex formulas as they appear in constraints. But again, certain aspects of the formula could be emphasized in a graphical way.

3 Relationship between TROLL light and UML

3.1 General overview

The above considerations are now made more precise. Our approach as depicted in Fig. 2 starts from the specification language TROLL light [GCH93, CGH95, HCG94], a language with which we have worked now for quite a while and for which (among other things) an algebraic semantics [GH95] is known. Related object specification languages are TROLL [JSHS91], TELOS [KMSJ88], Mondel [BBE$^+$90], CMSL [Wie91], ALBERT [DDP93], TQL [MT94] and RML [GMB94], among others. Two aspects (in the above sense of the word) of a TROLL light specification are the structural and dynamic system aspect. Other interesting aspects (e.g. an identifier aspect concerning identifier visibility) could be thought of. The structural system aspect (in OMT terminology the object model[1]) mainly concerns single system states whereas the dynamic system aspect (in OMT terms the dynamic model) explains how the system states evolve in time. Two possible views on the structure aspect could be a detailed single type view where one considers only one object type with all its structural attributes, and, in contrast to this, a clustered view on a complex structure model may abstract from the properties of single object types and may build clusters of object types belonging together by some given criteria. Only the relationships between such clusters may be represented in this clustered view. For the dynamic aspect different views may be established as well. In the detailed view we may look at a single object's evolution in all its aspects, for instance with completely given preconditions of allowed events. In contrast to this, in a substate view we may leave out preconditions and look at possible substates of a given section from the detailed view.

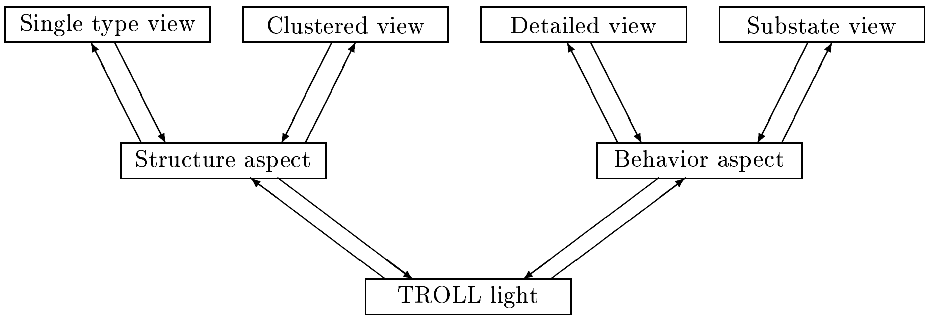

Figure 2. Two UML views on a TROLL light specification

As indicated in Fig. 3, for representing aspects semi-formally we choose the well-developing UML [BJR97] approach. In TROLL light specifications we have a signature part giving names to data types and object types (templates). Instances of data types (specified with a data type specification language like ACT ONE [EFH83], Extended ML [ST86], SPECTRUM [BFG^{+}93], or CASL [CoF97]) are data values whereas instances of object types are objects whose structural properties (its attributes) are given by data values and which can change in the course of time. Subobject relationships constitute part-of associations between objects, and events are the things happening to objects (its

[1] The use of the word model in this place is different from the usual use in the formal specification community. Here it refers to an aspect of a specification, not to a model in the sense of a formal interpretation (for instance an algebra) belonging to a signature and satisfying certain axioms.

operations). UML refers to this signature part of a TROLL light specification as static structure diagrams (OMT calls this the object model) where data types and classes (corresponding to templates) are given. TROLL light object-valued attributes realize UML associations, and TROLL light subobject relationships correspond to UML aggregations. For the dynamic part (in UML the state diagrams, in OMT the dynamic model) the TROLL light signature mentions the events corresponding to UML operations and UML events.

	TROLL light	UML
Signature	Data types	Data types in structure diagrams
	Templates	Classes in structure diagrams
	Attributes	Attributes, Associations in structure diagrams
	Subobjects	Aggregations in structure diagrams
	Events	Operations, Events in structure diagrams
Axioms	Valuation	Data types in structure and activity diagrams
	Constraints	Constraints in structure diagrams
	Derivation	Derived items in structure diagrams
	Interaction	Action transitions, Conditions in state diagrams
	Behavior	State diagrams

Figure 3. Comparison of TROLL light and UML

The axiom part of TROLL light together with the data type specification realize in the valuation rules UML activity diagrams. TROLL light constraints and derivations can be found in the UML static structure diagrams as constraints and as derived items. The dynamic aspect reflected in TROLL light by interaction rules representing action transitions and action preconditions are captured by details in the UML state diagrams. These are formally given in a TROLL light text as behavior patterns.

3.2 Drafting TROLL light concepts

Before presenting the details of our translation we explain central TROLL light concepts by means of an example describing a car rentals application scenario. One of the central object types is the booking template in Fig. 4. Booking objects use the data types String, Int, and Bool and the object types Customer and Car. They have object-valued attributes TheCustomer and TheCar of object sort customer and car, respectively. The convention is that (data and object) types start with upper-case letters whereas corresponding sorts begin with lower-case ones. Furthermore, there are data-valued attributes IsClosed, BookingCat, StartDay and EndDay giving a switch for whether there is still something to do with the booking or not, a booking category describing the car's desired equipment, and the required start and end day of the car usage. There are derived attributes (whose values are not stored but are calculated from other items) IsOpen and IsCurrent. The things to possibly change the state of a booking are the events (or operations), in this particular case a birth event createBooking, an event makeCurrent for actually assigning a car to a booking and an event to close the current booking.

```
TEMPLATE Booking
DATA TYPES String, Int, Bool;
TEMPLATES Customer, Car;
ATTRIBUTES
  TheCustomer : customer; TheCar : car; BookingCat : int;
  IsClosed : bool; StartDay, EndDay : int; -- YYYYMMDD
  DERIVED IsOpen : bool; DERIVED IsCurrent : bool;
EVENTS
  BIRTH createBooking(aCustomer : customer, aCat : int,
                      aStartDay : int, anEndDay : int);
  makeCurrent(aCar : car);
  close;
CONSTRAINTS
  1 <= BookingCat AND BookingCat <= 5;
  19900101 <= StartDay AND StartDay <= EndDay AND EndDay <= 20001231;
VALUATION
  [ createBooking(aCustomer,aCat,aStartDay,anEndDay) ]
      TheCustomer = aCustomer, BookingCat = aCat,
      StartDay = aStartDay, EndDay = anEndDay, IsClosed = FALSE;
  [ makeCurrent(aCar) ] TheCar = aCar;
  [ close ] TheCar = NIL, IsClosed = TRUE;
DERIVATION
  IsOpen = UNDEF(TheCar) AND NOT(IsClosed);
  IsCurrent = DEF(TheCar) AND NOT(IsClosed);
BEHAVIOR
  PROCESS Booking = ( createBooking -> BookingOpen );
  PROCESS BookingOpen = ( { BookingCat <= CarCat(aCar) }
                            makeCurrent(aCar) -> BookingCurrent );
  PROCESS BookingCurrent = ( close -> BookingClosed );
  PROCESS BookingClosed = ( );
END TEMPLATE
```

Figure 4. TROLL light Booking template

The possible object states are restricted in the constraints section. For demonstration purposes, we have modeled the booking category and the dates as integer values. A more sophisticated choice would be to introduce appropriate data types for these items. But nevertheless in both cases we need to require somewhere that the start day lies before (or is equal to) the end day of the booking. One place to do this is the constraints section described here, another choice would be a precondition in the behavior pattern to be described below. The valuations determine how the events (if they occur) effect the attribute values. The derivation section gives rules for computing the derived attribute values from other given things. In contrast to the simple derivations here, quite complex rules are also allowed in this place. For example, the Company template as given in Fig. 5 computes in the DeliverCarsMenu derived attribute of sort bag(tuple(int,int,bag(tuple(string,int)))) all open booking numbers and booking categories together with the collection of all free car numbers and car categories which fit to the booking. Last, as again shown in Fig. 4, the possible occurrence of events is described in a CSP-like notation in the behavior patterns. There is always one process with the name of the template to be defined which is the starting process, so here a booking's life begins with a createBooking event; after that the makeCurrent event is allowed (provided that the category of the given car fits the category of the booking); at last the event close may appear leading to a state where no further events are possible, but observations on the attributes can still be made. This Booking template example, however, does not mention the subobjects and the interaction section of a template.

In the subobjects section it is possible to introduce non-sharable components of the current object (sharing is possible via object-valued attributes). For example, the Company template as given in Fig. 5 describes the structure of the overall system object containing all other objects. Therefore, this template mentions the customers, cars and bookings as its subobjects. Regarding object identification, from the Company point of view a booking is identified by a booking number (given as an integer number parameter for the subobject symbol Bookings). From a conceptual point of view, the subobject symbol Bookings gives a means for accessing booking objects from company objects. From a technical point of view, the symbol Bookings can be looked at and used in three different ways: It can be used as (1) an infinite array (notation: `Bookings(BookingNo : int): booking`) with an integer index giving for each integer a booking object (in each state there is only a finite number of booking objects different from the undefined booking), (2) a map (notation: `Bookings : map(BookingNo : int, Booking : booking)`) from integers to bookings, or (3) a set of tuples (notation: `Bookings : set(tuple(BookingNo : int, Booking : booking))`) with two components of sort integer and booking and with the additional constraint that the tuple set also establishes a function (`FORALL (B1, B2 IN Bookings) BookingNo(B1) = BookingNo(B2) IMPLIES Booking(B1) = Booking(B2)`).

In the interaction section, communication patterns between objects are specified. For instance, it is possible to require the simultaneous occurrence of an event in a component subobject and an event in the enclosing aggregated object. In the Company template, the deliverCar event is, for instance, synchronized by interaction rules with the pickUp event of a corresponding car object and a makeCurrent event of a corresponding Booking object.

```
TEMPLATE Company
DATA TYPES String, Int;
TEMPLATES  Customer, Car, Booking;
SUBOBJECTS Customers(CustomerNo : int) : customer;
           Cars(CarNo : string) : car;
           Bookings(BookingNo : int) : booking;
ATTRIBUTES DeliverCarsMenu:bag(tuple(int,int,bag(tuple(string,int))));
...

EVENTS deliverCar(aBookingNo : int, aCarNo : string);
...

DERIVATION DeliverCarsMenu =
  SELECT BookingNo(B), BookingCat(Booking(B)),
    ( SELECT CarNo(Car(C)), CarCat(Car(C))
      FROM   C IN Cars
      WHERE  Available(Car(C)) AND
             CarCat(Car(C)) >= BookingCat(Booking(B)) )
  FROM   B IN Bookings
  WHERE  StartDay(Booking(B)) = Today AND IsOpen(Booking(B));
...

INTERACTION deliverCar(aBookingNo,aCarNo) >>
  Cars(aCarNo).pickUp,
  Bookings(aBookingNo).makeCurrent(Cars(aCarNo));
...

END TEMPLATE
```

Figure 5. Section of TROLL light Company template

3.3 Representing TROLL light descriptions with UML

In Fig. 6 and Fig. 7 we have displayed the UML representation of a structure model corresponding to the attributes, subobjects, and events sections of appropriate TROLL light templates for our scenario. Templates together with their data-valued attributes and events are pictured as rectangles. Associations (object-valued attributes and subobject relationships) between object types are denoted as lines. As displayed by the diamond, all Customer, Booking, and Car objects are aggregates (subobjects) of an object[2] belonging to template Company. Multiplicity is denoted as adornments at association edges. For example, the stars * close to Customer, Booking and Car denote that a company can consist of many such objects whereas the adornments 1 close to the diamonds near company expresses that each customer, booking, and car belongs to exactly one company. Also the references of a booking to its customer and car are given. In general, multiplicity is denoted by sets of integer intervals of the form low..high. The adornments * and 1 are abbreviations for 0..* and 1..1 where the star * expresses an unbounded cardinality.

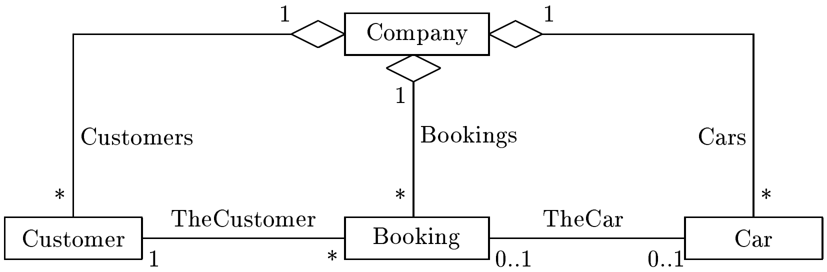

Figure 6. Translation of templates, subobjects and attributes

The dynamic aspect is visualized by the statechart-like diagram in Fig. 8. Here, the behavior patterns are presented. For example, the process description of Company is given by the upper left side of the diagram. States correspond to process names and transitions to event occurrences. For reasons of simplicity, the rather involved event preconditions are not displayed. They could be displayed as a separate view concentrating on single templates as this is shown in Fig. 9 for the Booking template. The interaction rules could be shown as additional adornments of edges like deliverCar / { pickUp, makeCurrent } because the

[2] This object may be viewed as "the" system object in the sense that all other objects appear as components respectively subobjects of this distinguished object. In the TROLL light specification, however, the company template is just a normal template. But its role as the system object template becomes evident when the TROLL light animation system is invoked by telling it that the session should start with exactly one object of sort company.

Booking
BookingCat : int
StartDay, EndDay : int
IsClosed, IsOpen, IsCurrent : bool
createBooking(aCustomer,aCat,aStartDay,anEndDay)
makeCurrent(aCar)
close

Figure 7. Translation of TROLL light attributes and events

event deliverCar in the DeliverCars state calls for event pickUp in Car and for event makeCurrent in Booking. The general possibility of concurrent event occurrences is represented as an orthogonal composition of and-states [HG96].

Please note, that not all parts in all aspects are visualized. We think for instance complex formulas as they appear in constraints or derivation rules should still be given as texts in linear form only. For example, the derived attribute DeliverCarsMenu with its rather complex definition is easier to understand when presented as a text.

3.4 General translation

In Fig. 10 we show the shape of a TROLL light specification with its component items. The corresponding UML picture for the signature part of a TROLL light specification is displayed in Fig. 11 as the structure model. Template (object type) names like $t, t', t'_1, ..., t'_n$, data-valued attributes as $a : d$, and events (operations) like $e(t'_1, ..., t'_m)$ are pictured within a class rectangle. Complex, object-valued attributes (denoted in TROLL light via attribute sort expressions involving set, list, bag, or tuple constructors) roughly correspond to UML associations given as Entity-Relationship diamonds.[3] Roughly here means that, in TROLL light we could, for example, have $a' : set(tuple(t_1, list(t_2)))$ or $a' : list(set(tuple(t_1, t_2)))$ but both would be represented in the same way as an UML structure diagram. Component relationships (in TROLL light terminology subobjects, in UML terms aggregations) are pictured as special associations with a small diamond at the side of aggregate. Thus, the subobject relation $s : t'$ is pictured as an aggregation characterized by a diamond. Again, the textual TROLL light description is able to give more subtle details than UML. For example, subobject symbols in TROLL light can have object-valued parameters (indicating existential dependence) that cannot be represented in UML.

[3] Diamonds are chosen in the case of three or more participating classes whereas binary links would be shown as direct lines.

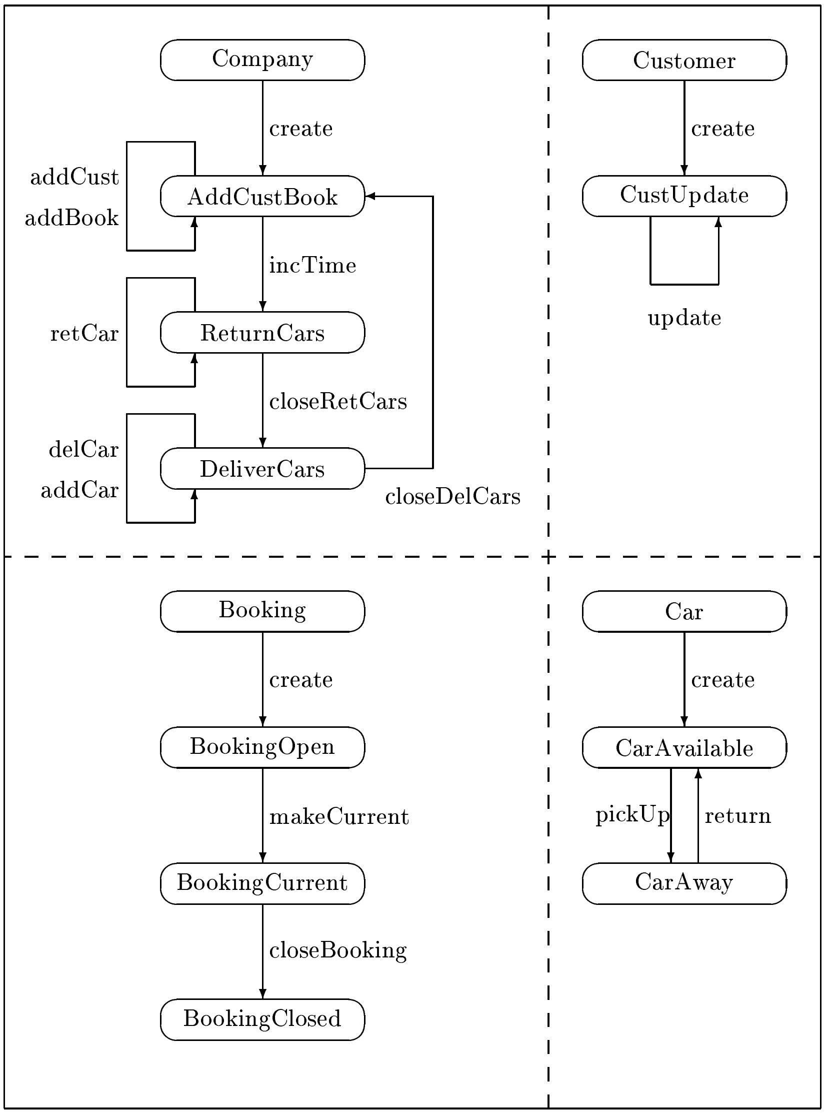

Figure 8. Translation of TROLL light behavior patterns

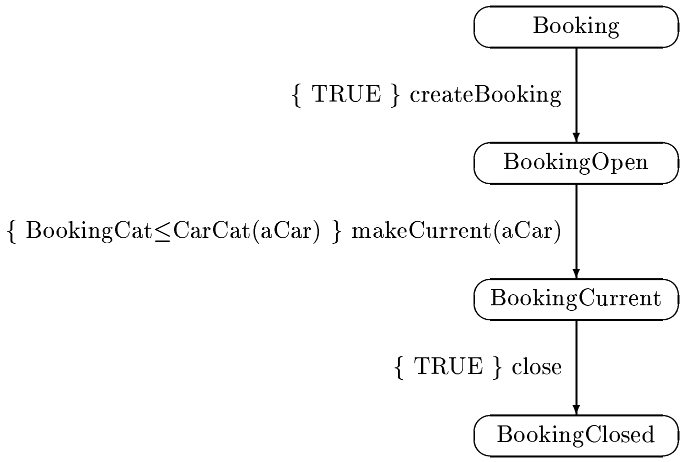

Figure 9. Translation of TROLL light Booking behavior

The translation does not touch the TROLL light constraints (where states are restricted by a formula φ) and derivation sections (where an attribute a'' is specified to have always the value of a term τ) which have textual representations in the UML diagrams.

`TEMPLATE`	t
`DATA TYPES`	d
`TEMPLATES`	$t_1, ..., t_n, t', t'_1, ..., t'_m$
`ATTRIBUTES`	$a : d$
	$a' : expr'(t_1, ..., t_n)$
`SUBOBJECTS`	$s : t'$
`EVENTS`	$e(p_1 : t'_1, ..., p_m : t'_m)$
`CONSTRAINTS`	φ
`DERIVATION`	$a'' = \tau$
`VALUATION`	$[e(p_1, ..., p_m)]\ a = expr(p_1, ..., p_m)$
`INTERACTION`	$e >> e'$
`BEHAVIOR`	`PROCESS` $x\ =\ (\ \{c\}\ e\ \rightarrow\ y\ )$
`END TEMPLATE`	

Figure 10. Important sections of TROLL light specifications

The behavioral model representing the interaction and behavior part is given in Fig. 12. Process names correspond to states and process transitions to event occurrences (eventually restricted by a precondition). Thus, in the UML picture we have nodes x and y and an edge labeled with the event e causing a transition for the given TROLL light process. If possible, we display in the same

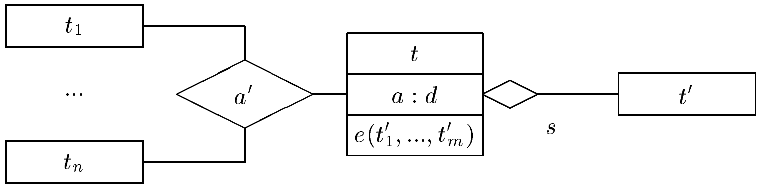

Figure 11. Translation of TROLL light signatures

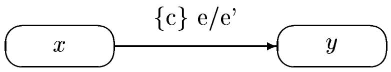

Figure 12. Translation of TROLL light behavior

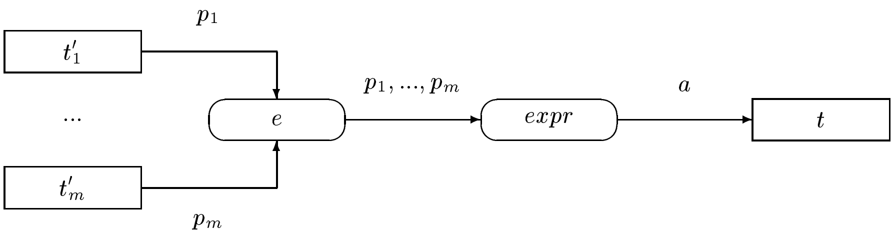

Figure 13. Translation of TROLL light valuation

diagram the interaction calling mechanism $e >> e'$ by stating the called event after the calling one with a slash as the separator (e/e' in the diagram). This is simple in the case that there are no preconditions. If there are more involved preconditions, then the behavior diagram must distinguish between the different cases by introducing corresponding alternative edges. Without going into details (due to space limitations) we mention that alternatively sequence diagrams can be used to represent event calling. The valuation axioms are reflected in UML by the activity diagrams indicated in Fig. 13. This diagram expresses that (parameter) objects p_i of sort t'_i flow into the event e (classified as an activity). Event e collects their values and passes them to the expression $expr$ of the valuation rule (also classified as an activity) which in turn hands the result of $expr(p_1, ..., p_m)$ to an object of sort t by updating the attribute a. Additionally, the corresponding data type specification (especially with its corresponding function definitions) and the interaction rules will give rise to more details for the activity diagrams.

4 Conclusion

We have presented a new approach for relating semi-formal and formal descriptions of software systems. In this approach we have chosen the object description language TROLL light for the formal part and the UML technique for the semi-formal part. There is a strong relation between development of a formal specification by means of semi-formal specifications and prototyping formal specifications for instance with an animation tool, e.g. the one presented in [RG97]. We think both approaches, i.e. prototyping and representing formal specifications by diagrams, are indispensable and complementary support mechanisms for achieving consistent and complete formal specifications of software systems.

We think it is relatively easy to derive corresponding UML diagrams from TROLL light specifications, whereas it is not that obvious to go from diagrams to formal specifications. At least one will have a certain degree of design freedom in doing this step, for instance in translating UML associations to object type attributes. In general it is an unsolved question how an already well-developed set of semi-formal and formal specifications can be modified consistently after the semi-formal specification has been changed.

As a last point for future work we mention the integration of what we call clustering techniques for structural [FM86, HZ90, RS92] and behavioral aspects [HG96] as a further abstraction mechanism. In larger designs one needs mechanisms for building clusters of design parts in order to achieve a degree of detail that can be understood. A cluster representing a structural or dynamic system aspect is considered to be, for example, a higher level node representing a collection of nodes on the lower level. The benefit of abstracting several nodes to one abstract node lies of course in the improved clearness of the underlying diagram. This problem however is also unsolved on the semi-formal UML level.

References

BBE+90. G. v. Bochmann, M. Barbeau, M. Erradi, L. Lecomte, P. Mondain-Monval, and N. Williams. Mondel: An Object-Oriented Specification Language. Département d'Informatique et de Recherche Opérationelle, Publication 748, Université de Montréal, 1990.

BFG+93. M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hussmann, D. Nazareth, F. Regensburger, O. Slotosch, and K. Stølen. The Requirement and Design Specification Language SPECTRUM — An Informal Introduction (Version 1.0). Technical Report TUM I9311-12, TU München, 1993.

BHH+97. Ruth Breu, Ursula Hinkel, Christoph Hofmann, Cornel Klein, Barbara Paech, Bernhard Rumpe, and Veronika Thurner. Towards a Formalization of the Unified Modeling Language. In Mehmet Aksit and Satoshi Matsuoka, editors, *Proc. 11th European Conf. Object-Oriented Programming (ECOOP'97)*, pages 344–366. Springer, Berlin, LNCS 1241, 1997.

BJR97. G. Booch, I. Jacobson, and J. Rumbaugh. *UML Summary (Version 1.0)*. Rational Corporation, Santa Clara, 1997. http://www.rational.com.

CGH95. S. Conrad, M. Gogolla, and R. Herzig. Safe Derivations in Object Hierarchies. In D. Patel, Y. Sun, and S. Patel, editors, *Int. Conf. Object-Oriented Information Systems (OOIS'94)*, pages 306–319. Springer, London, 1995.

CoF97. CoFI Task Group on Language Design. CASL - The CoFI Algebraic Specification Language. Technical Report, DAIMI, Aarhus, Danmark, 1997.

DDP93. E. Dubois, P. Du Bois, and M. Petit. O-O Requirements Analysis: An Agent Perspective. In O.M. Nierstrasz, editor, *Proc. European Conf. on Object-Oriented Programming (ECOOP'93)*, pages 458–481. Springer, Berlin, LNCS 707, 1993.

EFH83. H. Ehrig, W. Fey, and H. Hansen. ACT ONE: An Algebraic Specification Language with Two Levels of Semantics. Technical Report 83-03, Technische Universität Berlin, 1983.

FM86. P. Feldman and D. Miller. Entity Model Clustering: Structuring a Data Model by Abstraction. *Computer Journal*, 29(4):348–360, August 1986.

GCH93. M. Gogolla, S. Conrad, and R. Herzig. Sketching Concepts and Computational Model of TROLL light. In A. Miola, editor, *Proc. 3rd Int. Conf. Design and Implementation of Symbolic Computation Systems (DISCO)*, pages 17–32. Springer, Berlin, LNCS 722, 1993.

GH95. M. Gogolla and R. Herzig. An Algebraic Semantics for the Object Specification Language TROLL light. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Type Specification (WADT'94)*, pages 288–304. Springer, Berlin, LNCS 906, 1995.

GMB94. S. Greenspan, J. Mylopoulos, and A. Borgida. On Formal Requirements Modeling Languages: RML Revisited. In B. Fadini, editor, *Proc. 16th Int. Conf. on Software Engineering (ICSE'94)*, pages 135–148. IEEE Computer Society Press, 1994.

HCG94. R. Herzig, S. Conrad, and M. Gogolla. Compositional Description of Object Communities with TROLL light. In C. Chrisment, editor, *Proc. Basque Int. Workshop on Information Technology (BIWIT'94): Information Systems Design and Hypermedia*, pages 183–194. Cépaduès-Éditions, Toulouse, 1994.

HG96. D. Harel and E. Gery. Executable Object Modeling with Statecharts. In *18th Int. Conf. Software Engineering*, pages 246–257. Springer Verlag, Berlin, 1996.

HZ90. S. Huffman and R. V. Zoeller. A Rule-Based System Tool for Automated ER Model Clustering. In F. H. Lochovsky, editor, *Proc. 8th Int. Conf. Entity-Relationship Approach*, pages 221–236. Elsevier Science Publishers, 1990.

JSHS91. R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. Object-Oriented Specification of Information Systems: The TROLL Language. Informatik-Bericht 91–04, Technische Universität Braunschweig, 1991.

JWH+94. R. Jungclaus, R. J. Wieringa, P. Hartel, G. Saake, and T. Hartmann. Combining TROLL with the Object Modeling Technique. In B. Wolfinger, editor, *Innovationen bei Rechen- und Kommunikationssystemen. GI-Fachgespräch FG 1: Integration von semi-formalen und formalen Methoden für die Spezifikation von Software*, pages 35–42. Informatik aktuell, Springer, Berlin, 1994.

KMSJ88. M. Koubarakis, J. Mylopoulos, M. Stanley, and M. Jarke. TELOS: A Knowledge Representation Language for Requirements Modelling. Technical Report CSRI-222, University of Toronto, 1988.

LR97. M. Larosa and G. Reggio. A Graphic Notation for Formal Specifications of Dynamic Systems. In P. Lucas, editor, *Proc. 4th Int. Symposium Formal Methods Europe (FME'97)*. Springer, LNCS, 1997.

MT94. M. Missikoff and M. Toiati. MOSAICO - A System for Conceptual Modeling and Rapid Prototyping of Object-Oriented Database Application. *SIGMOD Record*, 23(2):508–519, 1994.

RBP+91. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs (NJ), 1991.

RG97. M. Richters and M. Gogolla. A Web-based Animator for Object Specifications in a Persistent Environment. In M. Bidoit and M. Dauchet, editors, *Proc. 7th Int. Conf. Theory and Practice of Software Development (TAPSOFT'97)*, pages 867–870. Springer, Berlin, LNCS 1214, 1997.

RS92. O. Rauh and E. Stickel. Entity Tree Clustering - A Method for Simplifying ER Designs. In *Proc. Inf. Conf Entity-Relationship Approach (ERA'92)*, 1992.

RS95. J. Ramos and A. Sernadas. A Brief Introduction to GNOME. Research Report, Section of Computer Science, Department of Mathematics, Instituto Superior Técnico, 1096 Lisboa, Portugal, 1995.

ST86. D.T. Sannella and A. Tarlecki. Extended ML: An Institution-Independent Framework for Formal Program Development. In *Proc. Workshop on Category Theory and Computer Programming*, pages 364–389. Springer, LNCS 240, 1986.

Wie91. R.J. Wieringa. Equational Specification of Dynamic Objects. In R.A. Meersman, W. Kent, and S. Khosla, editors, *Object-Oriented Databases: Analysis, Design & Construction (DS-4), Proc. IFIP WG2.6 Working Conf.*, pages 415–438. North-Holland, Amsterdam, 1991.