Recent Trends in Data Type Specification (WADT'94),E. Astesiano,
G. Reggio, A. Tarlecki (Eds.), Springer, Berlin, LNCS, pp. 290-306 (1995)

An Algebraic Semantics for the Object Specification Language TROLL *light*[*]

Martin Gogolla & Rudolf Herzig

Universität Bremen, FB Mathematik und Informatik
AG Datenbanksysteme, Postfach 330440, D-28334 Bremen, Germany
e-mail: gogolla@informatik.uni-bremen.de

Abstract. Within the KORSO project we have developed the object specification language TROLL *light* which allows to describe the part of the world to be modeled as a community of concurrently existing and communicating objects. Recently, we have worked out the basic notions of a pure algebraic semantics for our language. The main underlying idea is to present a transition system where the states represent the states of the specified information system, and state transitions are caused by the occurrence of finite sets of events. This semantics is formulated by representing states and state transitions as algebras. The various constructs of TROLL *light* are unified to general axioms restricting the possible interpretations for TROLL *light* object descriptions.

1 Introduction

The information system development process can be divided into two phases: The requirements and the design engineering phase. At the end of the requirements engineering phase, a first formal description of the system to be developed should be the "contract" for further development. Therefore, the task of delivering this first formal specification is of central concern.

One language designed for this task is the object description language TROLL *light* [CGH92, GCH93, HCG94], a dialect of OBLOG [SSE87] and TROLL [JSHS91]. Other recent object specification languages are ABEL [DO91], CMSL [Wie91], GLIDER [CJO94], MONDEL [BBE$^+$90], OCS [AAR94], OS [Bre91], and Π [Gab93]. Our language TROLL *light* incorporates ideas from algebraic specification [EM85, EM90, Wir90], semantic data models [HK87, PM88], and process theory [Hoa85, Hen88, Mil89]. It is a language for describing static and dynamic properties of objects. This is achieved by offering language features to specify object structure as well as object behavior. The main advantage of following the object paradigm is the fact that all relevant information concerning one object can be found within one single unit and is not distributed over a variety of locations.

[*] Work reported here has been partially supported by the CEC under Grant No. 6112 (COMPASS).

We explain the essential characteristics of TROLL *light* by means of an example. This small example will be used throughout in order to explain all details of our approach. In Fig. 1 binary trees, where the nodes carry boolean information, are

```
TEMPLATE Node
  DATA TYPES  Bool;
  ATTRIBUTES  Content:bool;
  SUBOBJECTS  Left, Right:node;
  EVENTS      BIRTH create;
                    createLeft;
                    createRight;
                    update(Content:bool);
              DEATH destroy;
  VALUATION   [ create ] Content=true;
              [ update(B) ] Content=B;
  INTERACTION createLeft >> Left.create;
              createRight >> Right.create;
  BEHAVIOR    PROCESS Node =
                ( create -> NodeLife );
              PROCESS NodeLife =
                ( createLeft, createRight, update -> NodeLife |
                  { UNDEF(Left) AND UNDEF(Right) }
                    destroy -> Node );
END TEMPLATE
```

Fig. 1. TROLL *light* example template

specified as object types (or templates as object types are called in TROLL and TROLL *light*).

`DATA TYPES`**:** In TROLL *light* data types are employed for various purposes. Data types are assumed to be specified with a data type specification language. Their signature is made known in a template with the `DATA TYPES` section.

`ATTRIBUTES`**:** The `Node` template uses the data type `Bool` in order to define one attribute `Content` of data sort `bool`.

`SUBOBJECTS`**:** For templates (or object types) we employ the same naming convention as for data types. Template names are written capitalized, and each template (`Node`) induces a corresponding object sort written exactly as the template but with a starting lower case letter (`node`). A `Node` object is allowed to have two local subobjects, namely `Left` and `Right`, of sort `node`.

`EVENTS`**:** The things which can happen to `Node` objects are called events. `Node` objects can be created, their left and right subnodes can be created, they can be updated, and they can be destroyed.

`VALUATION`**:** Valuation rules serve to define the effect of events on attributes. They specify the value an attribute has after the occurrence of an event.

`INTERACTION`**:** Interaction rules are used to describe the synchronization of events. For instance, whenever the `createLeft` event occurs in the parent object, the `create` event must occur for the left subobject. New objects can only be created as subobjects of existing objects by this mechanism which is called birth event calling.

`BEHAVIOR:` Possible life cycles, i.e., sequences of allowed events, are specified in the behavior part. For a given template there is always a process with the same name which is the initial process for objects belonging to the template. An objects' life starts with a `BIRTH` event and possibly ends with a `DEATH` event, which in this case is only allowed to occur when there are no subobjects. In between, the events `createLeft`, `createRight`, and `update` can occur, but due to the interaction rule, for instance, `createLeft` can only occur if the called event `create` fits into the subobjects' life cycle.

Above, we have shown a simple example of a TROLL *light* template for introductory purposes. For more sophisticated applications, we have to refer to [CGH92, GCH93, HCG94, GHC$^+$94]. There, you find for instance parameterized attributes and subobject symbols, complex attributes, and non-trivial event synchronization.

An attractive specification language on its own is not enough. It must be completed by specification tools. Therefore TROLL *light* comes along with a development environment offering special tools for verification and validation purposes. In order to support verification, the TROLL *light proof support system* solves verification tasks given by the user. It checks whether the desired properties are fulfilled by the specified TROLL *light* object descriptions. The TROLL *light animator* is designed to simulate the behavior of a specified object community. By this, the informal view of the real-world fragment to be modeled is validated against the current specification. Details of the development environment can be found in [GVH$^+$94].

The remainder of this paper is organized as follows. In Sect. 2 we explain the basic ingredients of our approach to object specification, namely object signatures and models for such signatures. Section 3 introduces axioms in order to restrict the class of models for object signatures and explains how a standard model for TROLL *light* specifications is obtained. In Sect. 4 we shortly compare our approach with other proposals which appeared in recent literature. Section 5 closes the paper with some concluding remarks and mentions future work to be done.

2 Signatures and Models

Here, we introduce the main components of our approach. First, we introduce data signatures and data algebras. Second, signatures for object communities including attributes and events are proposed. These object signatures are reduced to conventional algebraic signatures. Doing this, we can afterwards define models, namely object communities, as special algebraic transition systems.

Definition 1. A **data signature** $D\Sigma = (DS, \Omega)$ consists of a set of data sorts DS and a family of sets of operation symbols $\Omega = \langle \Omega_{w,ds} \rangle_{w \in DS^*, ds \in DS}$. A **$D\Sigma$-model** $M_{D\Sigma}$, i.e., an algebra, is given by a family of sets $\langle M_{ds} \rangle_{ds \in DS}$ and a family of functions $\langle F_{w,ds} \rangle_{w \in DS^*, ds \in DS}$ such that for $\omega : ds_1 \times ... \times ds_n \to ds \in \Omega$ domain and range of ω_M are determined by $\omega_M : M_{ds_1} \times ... \times M_{ds_n} \to M_{ds}$. The category of all $D\Sigma$-models is denoted by $ALG_{D\Sigma}$. We assume one data signature $D\Sigma$ and one data algebra DA to be fixed.

Example 2 (Data signature and data algebra). The above TROLL *light* template uses the data type *Bool* with the following data signature.

$\quad$ *data sorts bool*

$\quad$ *operations* $false, true, bottom :\to bool$

The data algebra for this signature is the term algebra $T_{D\Sigma}$. As an alternative to completing signatures and algebras with bottom elements, we could use partial algebras. However, we want to be as close as possible to our implementation [HG94b] which allows bottom expressions, for instance, as user input representing information that is unknown.

Definition 3. Let a data signature $D\Sigma$ be given. An **object signature** $O\Sigma = (OS, A, E)$ over the data signature $D\Sigma$ consists of a set of object sorts OS, a family of sets of attribute symbols $A = \langle A_{os,w,s} \rangle_{os \in OS, w \in S^*, s \in S}$, and a family of sets of event symbols $E = \langle E_{os,w} \rangle_{os \in OS, w \in S^*}$, where S refers to the union of DS and OS, i.e., $S := DS \cup OS$.

Example 4 (Object signature). The above template did not mention the object sort for which the attributes and events of the template are defined. This is now done explicitly in the corresponding object signature. For the time being, we treat TROLL *light* subobject symbols like attribute symbols. The distinction between both will become clear later in Requirement 25 concerning the interpretation of subobject symbols.

object sorts *node*	
attributes	$left, right : node \to node$
	$content : node \to bool$
events	$create : node$
	$createLeft, createRight : node$
	$update : node \times bool$
	$destroy : node$

Definition 5. Let an object signature $O\Sigma$ be given. The **attribute signature** $A\Sigma$ induced by $O\Sigma$ consists of S, i.e., the union of data and object sorts, and the union of operation and attribute symbols, i.e., $A\Sigma = (DS \cup OS, \Omega \cup A)$.

As already proposed in [Gog89], attribute algebras belonging to attribute signatures are used to describe the states of a system. Thus a single algebra represents the system state in one instant of time.

Example 6 (Attribute signature and attribute algebra). The attribute signature for our running example has the following shape.

$$\begin{array}{ll} \mathit{sorts} & \mathit{bool}, \mathit{node} \\ \mathit{operations} & \mathit{false}, \mathit{true}, \mathit{bottom} : \to \mathit{bool} \\ & \mathit{left}, \mathit{right} : \mathit{node} \to \mathit{node} \\ & \mathit{content} : \mathit{node} \to \mathit{bool} \end{array}$$

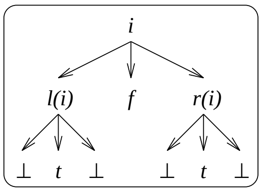

Fig. 2. Example of an attribute algebra $M_{A\Sigma}$

In Fig. 2 we graphically show one attribute algebra $M_{A\Sigma}$ whose data part is required to be isomorphic to the term algebra of the data signature, i.e., $U_{A\Sigma \to D\Sigma}(M_{A\Sigma}) = T_{D\Sigma}$[2]. The algebra $M_{A\Sigma}$ has three "proper" node objects $(i, l(i), r(i))$ and one special object representing undefined ($\perp$). The functions in $M_{A\Sigma}$ are understood to be defined as described in the table in Fig. 3. In the graphical representation, left arrows represent the function $left_{M_{A\Sigma}}$, middle arrows the function $content_{M_{A\Sigma}}$, right arrows the function $right_{M_{A\Sigma}}$, and the undefined object $\perp$ is duplicated as it is needed.

$n \in M_{A\Sigma,node}$	$left_{M_{A\Sigma}}(n)$	$right_{M_{A\Sigma}}(n)$	$content_{M_{A\Sigma}}(n)$
i	$l(i)$	$r(i)$	$false$
$l(i)$	$\perp$	$\perp$	$true$
$r(i)$	$\perp$	$\perp$	$true$
$\perp$	$\perp$	$\perp$	$bottom$

Fig. 3. Definition of functions in $M_{A\Sigma}$

Definition 7. Let an object signature $O\Sigma$ be given. The **event signature** $E\Sigma$ induced by $O\Sigma$ consists of the attribute signature extended by one special sort symbol $\hat{e}$ for events and an appropriate family of event symbols $\hat{E}$, i.e., $E\Sigma = (DS \cup OS \cup \{\hat{e}\}, \Omega \cup A \cup \hat{E})$ with $\hat{E} = \langle \hat{E}_{os,w,\hat{e}} \rangle_{os \in OS, w \in S^*}$ and $\hat{E}_{os,w,\hat{e}} := E_{os,w}$.

Algebras of event signatures describe possible transitions caused by the set of events in the interpretation of sort $\hat{e}$.

Example 8 (Event signature and event algebra). For the running example the event signature given in Fig. 4 shows up.

[2] $U_{A\Sigma \to D\Sigma}$ denotes the forgetful functor from $A\Sigma$-algebras to $D\Sigma$-algebras.

sorts	$bool, node, \hat{e}$	
operations	$false, true, bottom :\to bool$	$(f, t, \perp)$
	$left, right : node \to node$	(l, r)
	$content : node \to bool$	(ct)
	$create : node \to \hat{e}$	(c)
	$createLeft, createRight : node \to \hat{e}$	(cL, cR)
	$update : node \times bool \to \hat{e}$	(u)
	$destroy : node \to \hat{e}$	(d)

Fig. 4. Example of an event signature

In the following we abbreviate the operation names as indicated on the right hand side above. To give an example for an event algebra $M_{E\Sigma}$ we require that the attribute part of $M_{E\Sigma}$ is isomorphic to the attribute algebra from Example 6, i.e., $U_{E\Sigma\to A\Sigma}(M_{E\Sigma}) = M_{A\Sigma}$. This event algebra $M_{E\Sigma}$ is intended to be an algebra describing possible events occurring in the attribute algebra of Example 6.
The more interesting part of $M_{E\Sigma}$ is the interpretation of $\hat{e}$. Intuitively, this event algebra requires three events to happen, namely three updates for the "proper" objects or more formally $M_{\hat{e}} = \{u(i,t), u(l(i), f), u(r(i), f), \perp\}$. The operations are defined as follows: $u_{M_{E\Sigma}}(i,t) = u(i,t)$, $u_{M_{E\Sigma}}(l(i), f) = u(l(i), f)$, $u_{M_{E\Sigma}}(r(i), f) = u(r(i), f)$; all other applications of event symbols yield $\perp$. For instance, events which are in this situation not allowed in the life cycles (like $c(i)$) or events not changing an attribute (like $u(i, f)$) evaluate to bottom: $c_{M_{E\Sigma}}(i) = \perp$ and $u_{M_{E\Sigma}}(i, f) = \perp$. Thus, the exception value $\perp$ indicates that an event does not happen.
In this example (and in the ones to follow) we assume the events in the interpretation of $\hat{e}$ to be freely generated over an adequate attribute algebra. But this is not a general restriction.

The overall syntactic situation is depicted in Fig. 5. The signatures $D\Sigma$, $A\Sigma$, and $E\Sigma$ are all signatures in the classical sense. Thus, object signatures with event symbols have been reduced to classical signatures.
We now come to our central definition, namely the definition of a model for an object signature. Because in such models different objects co-exist and communicate with each other, we call such models object communities.

Definition 9. Let an object signature $O\Sigma$ over $D\Sigma$ together with the induced attribute signature $A\Sigma$ and event signature $E\Sigma$ be given. An **$O\Sigma$-model** $M_{O\Sigma}$, i.e., an **object community**, is given by a relation

$$M_{O\Sigma} \subseteq ALG_{A\Sigma} \times ALG_{E\Sigma} \times ALG_{A\Sigma}.$$

The data part of triples $\langle A_L, \hat{A}, A_R\rangle$ in $M_{O\Sigma}$ is required to coincide with the fixed data algebra DA, i.e., $U_{A\Sigma\to D\Sigma}(A_L) = U_{E\Sigma\to D\Sigma}(\hat{A}) = U_{A\Sigma\to D\Sigma}(A_R) = DA$.

Intuitively, triples $\langle A_L, \hat{A}, A_R\rangle$ in $M_{O\Sigma}$ express that there is a state transition from attribute algebra A_L to attribute algebra A_R via the occurrence of the

$$
\begin{tikzcd}
 & D\Sigma = (DS, \Omega) \arrow[d, dash] \\
(OS, A, E) = O\Sigma \arrow[ur, dash] \arrow[r] \arrow[dr] & A\Sigma = (DS \cup OS, \Omega \cup A) \arrow[d, dash] \\
 & E\Sigma = (DS \cup OS \cup \{\hat{e}\}, \Omega \cup A \cup \hat{E})
\end{tikzcd}
$$

Fig. 5. Signatures for object communities

events in $\hat{A}$. Of course, triples $\langle A_L, \hat{A}, A_R \rangle$ in $M_{O\Sigma}$ can also be regarded as one algebra whose signature is the disjoint union of $A\Sigma$, $E\Sigma$, and $A\Sigma$.

Example 10 (Object community). In Fig. 6 we have sketched part of an object community for our running example. Each arrow stands for a triple $\langle A_L, \hat{A}, A_R \rangle$. Event algebras are depicted next to arrows by giving only the elements of $\hat{e}$ different from $\perp$. Let us explain some details a bit more. (1) The transition along $\{cR(i), c(r(i))\}$ stands for the event $cR(i)$ occurring in i and the create event $c(r(i))$ occurring in $r(i)$. (2) The transition along $\{d(l(i)), d(r(i))\}$ stands for the simultaneous occurrence of two delete events in the respective objects. Of course much more transitions than the depicted ones are possible.
A general remark on the usefulness of the algebraic properties of our approach might be in order here. In this example, we have decided that $cR(i)$ and $c(r(i))$ denote different elements in an event algebra. But our approach also allows to model the interaction rule more directly by giving an event algebra $M_{E\Sigma}$ for the transition where both $cR(i)$ and $c(r(i)$ evaluate to the same element, for instance to a quotient element: $cR_{M_{E\Sigma}}(i) = c_{M_{E\Sigma}}(r(i)) = \{cR(i), c(r(i))\}$. This reflects the fact that the occurrence of the two events is considered as an atomic unit from a conceptional point of view. It possible only by taking a general algebraic structure into consideration.
For the transition $\langle A_L, \hat{A}, A_R \rangle$ belonging to $\{u(i,t), u(l(i), f), u(r(i), f)\}$ it was the case that $U_{E\Sigma \to A\Sigma}(\hat{A}) = A_L$ was true. But in general, we cannot require that the $A\Sigma$-part of $\hat{A}$ coincides with A_L or A_R because in one transition some objects may vanish while others newly show up. Thus, it seems most likely that $\hat{A}$ includes all objects in A_L and A_R. Therefore, it makes sense to require that $\hat{A}_{node} = A_{L,node} \cup A_{R,node}$ holds. One can generalize this remark and could further restrict the definition of an object community $M_{O\Sigma}$ in the sense that $\hat{A}_{os} = A_{L,os} \cup A_{R,os}$ must be true for all triples in $M_{O\Sigma}$.

3 Algebraic Semantics of TROLL *light*

Our TROLL *light* example template does not mention all possibilities of the language. In general, templates show the structure given in Fig. 7.

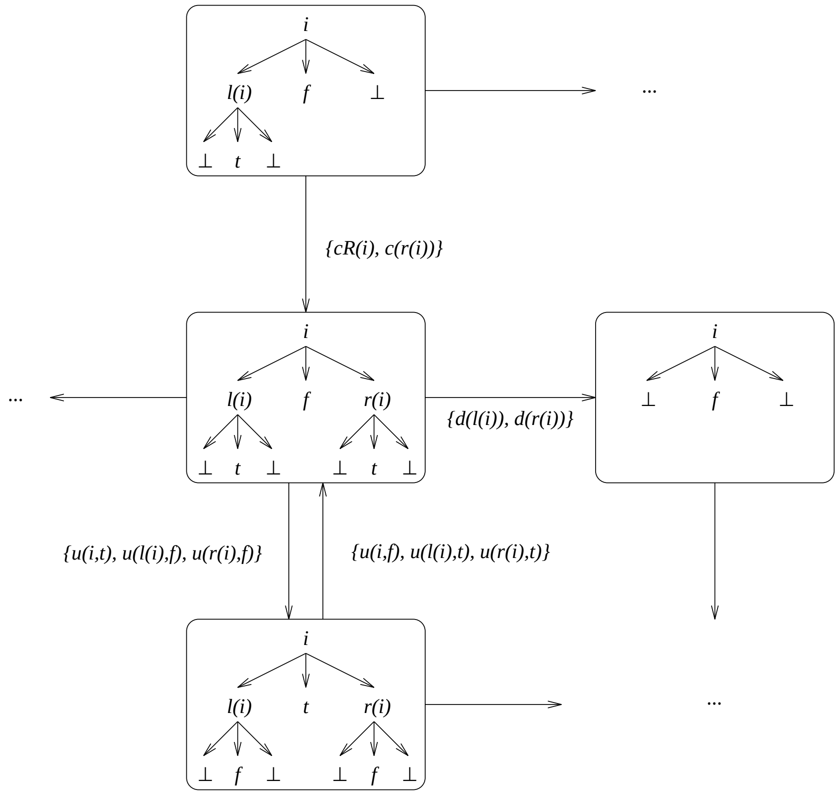

Fig. 6. Part of an example for a simple object community

`TEMPLATE` name of the template	
`DATA TYPES`	data types used in current template
`TEMPLATES`	other templates used in current template
`ATTRIBUTES`	slots for attributes
`SUBOBJECTS`	slots for subobjects
`EVENTS`	event generators
`CONSTRAINTS`	restricting conditions on object states
`DERIVATION`	rules for derived attributes
`VALUATION`	effect of event occurrences on attributes
`INTERACTION`	synchronization of events in different objects
`BEHAVIOR`	description of object behavior by a CSP-like process
`END TEMPLATE`	

Fig. 7. General structure of TROLL *light* templates

The TROLL *light* `ATTRIBUTES` and `SUBOBJECTS` determine the object signature and the TROLL *light* `EVENTS` the event signature. Thus, we have discussed the signature part, and that means we have finished the `DATA TYPES`, `TEMPLATES`, `ATTRIBUTES`, `SUBOBJECTS`, and `EVENTS` sections. We now consider the remaining sections corresponding to axioms over the object signature. For the following definitions let an object signature $O\Sigma$ and an object community $M_{O\Sigma}$ over $O\Sigma$ be given.

Definition 11. A **constraint** φ is a formula restricting the class of possible attribute algebras $ALG_{A\Sigma}$. A constraint is **valid** in $M_{O\Sigma}$ iff for all triples $\langle A_L, \hat{A}, A_R\rangle$ in $M_{O\Sigma}$ φ holds in A_L and A_R.
We do not require that φ holds in the intermediate algebra $U_{E\Sigma\to A\Sigma}(\hat{A})$. The details of the logic for the formulation of constraints — a variation of first-order predicate calculus tailored for information system purposes — can be found in [HG94a].

Example 12 (Constraint). As an example for our `Node` template consider the constraint that, if the subobjects exist, then the `Content` attribute values of the subobjects are not allowed to coincide.

```
(DEF(Left) AND DEF(Right)) => (Content(Left)<>Content(Right))
```

Definition 13. A **derivation rule** is of the form $\alpha = t_\alpha$ where α is an attribute and t_α is a term with the same sort as the attribute. A derivation rule is **valid** in $M_{O\Sigma}$ iff for all triples $\langle A_L, \hat{A}, A_R\rangle$ in $M_{O\Sigma}$ attribute α evaluates in A_L and A_R to the value of t_α.

Example 14 (Derivation rule). As an example for a `Node` template derivation rule consider an attribute `Count:nat` which should give for a node object the number of direct and indirect subobjects including the node object itself.

```
Count =
  ( IF DEF(Left)  THEN Count(Left)  ELSE 0 FI ) +
  ( IF DEF(Right) THEN Count(Right) ELSE 0 FI ) + 1
```

The above derivation rule is an example for a recursively defined derived attribute. Nevertheless, in this case the evaluation is well-defined because no cycles can be introduced with subobject relationships. But we will come to this point later in Example 26 after we have discussed the special interpretation of subobject symbols.

Definition 15. A **valuation rule** is of the form $\{\varphi\}[t_{\hat{e}}]\alpha = t_\alpha$, where φ is a formula, $t_{\hat{e}}$ is an event term, α is an attribute, and t_α is a term with the same sort as the attribute α. A valuation rule is **valid** in $M_{O\Sigma}$ iff for all triples $\langle A_L, \hat{A}, A_R\rangle$ in $M_{O\Sigma}$ the following is true: If the formula φ holds in A_L and the event $t_{\hat{e}}$ occurs in[3] $\hat{A}$, then the attribute α evalutes in A_R to the value which t_α had in A_L.

[3] We say an event e occurs in $\hat{A}$ if e evaluates to something different from $\perp$ in $\hat{A}$.

Example 16 (Valuation rule). Examples for valuation rules appeared in the introductory TROLL *light* template. For instance, the rule for the event `create` sets the `Content` attribute to `true`. Missing formulas φ are assumed to be *true*.

Definition 17. An **interaction rule** is of the form $\{\varphi\}\ t_{\hat{e}} >> t'_{\hat{e}}$ where φ is a formula, and $t_{\hat{e}}$ and $t'_{\hat{e}}$ are event terms. An interaction rule is **valid** in $M_{O\Sigma}$ iff for all triples $\langle A_L, \hat{A}, A_R\rangle$ in $M_{O\Sigma}$ the following holds: If φ is true in A_L and $t_{\hat{e}}$ occurs in $\hat{A}$, then $t'_{\hat{e}}$ must also occur in $\hat{A}$.

Example 18 (Interaction rule). Examples for interaction rules showed up in our running TROLL *light* example. For instance, the first interaction rule demanded, that whenever the `createLeft` event in the current object occurs, the `create` event must occur for the left subobject as well. Missing formulas φ are again assumed to be *true*.
The running example shows only interaction rules where birth events are involved. In principle, however, event calling between arbitrary events is possible.

A TROLL *light* object is always in one of a fixed set of process states. Therefore we have to extend for each template T the object signature by a special sort $processStateSort_T$ and a special attribute $processState_T : os_T \to processStateSort_T$ reflecting this current process state (os_T is the object sort induced by template T). Roughly speaking, the values this attribute can take are the process names mentioned in the template. Furthermore, the event symbols have to be splitted into *birth*, normal *life*, and *death* events according to the given TROLL *light* specification.

Definition 19. A **behavior definition** for a template T consists of a set of process definitions of the following form:
$\Pi_0 = (\{\varphi_1\}t_{\hat{e}_1} \to \Pi_1 \mid \{\varphi_2\}t_{\hat{e}_2} \to \Pi_2 \mid ... \mid \{\varphi_n\}t_{\hat{e}_n} \to \Pi_n)$
Here the Π_i's are process names, the φ_i's formulas, and the $t_{\hat{e}_i}$'s event terms. A behavior definition for template T is **valid** in $M_{O\Sigma}$ iff for all triples $\langle A_L, \hat{A}, A_R\rangle$ in $M_{O\Sigma}$ the following holds: If an event e of template T occurs in $\hat{A}$ for an object o of object sort os_T, then there is a process definition and an index j such that $t_{\hat{e}_j}$ evaluates in $\hat{A}$ to e, φ_j is true in A_L, $processState_T(o) = \Pi_0$ in A_L, and $processState_T(o) = \Pi_j$ in A_R. Furthermore, a birth event b for an object o is only allowed to occur in a transition $\langle A_L, \hat{A}, A_R\rangle$ if $o \notin A_L$, b occurs in $\hat{A}$, and $o \in A_R$ holds. A death event d for o is only allowed to occur if $o \in A_L$, d occurs in $\hat{A}$, $o \notin A_R$, and $o' \notin A_R$ holds for all subobjects[4] o' of o.

Example 20 (Behavior definition). The `BEHAVIOR` section of the running example required that a node's life starts with the `create` event. Afterwards `createLeft`, `createRight`, and `update` events follow as long as (in particular) interaction rules are obeyed. So, for example, a `createLeft` event is only allowed if the left subobject accepts the `create` event in its life cycle. The `destroy` event is only allowed to occur when there are no subobjects. Let us finally consider in Fig. 8 how the running example looks like if we present it in the syntax according to the previous definitions.

[4] The notion of subobject is made precise in Requirement 25.

$$
\begin{array}{ll}
valuation & \{true\}\ [create(Self)]\ Content(Self) = true \\
 & \{true\}\ [update(Self, B)]\ Content(Self) = B \\
interaction & \{true\}\ createLeft(Self)\ >>\ create(Left(Self)) \\
 & \{true\}\ createRight(Self)\ >>\ create(Right(Self)) \\
behavior & Node = (\{true\}\ create(Self) \to NodeLife\ ) \\
 & NodeLife = (\ \{true\}\ createLeft(Self) \to NodeLife\ | \\
 & \qquad\qquad\quad \{true\}\ createRight(Self) \to NodeLife\ | \\
 & \qquad\qquad\quad \{true\}\ update(Self, B) \to NodeLife\ | \\
 & \qquad\qquad\quad \{\neg Def(Left(Self)) \wedge \neg Def(Right(Self))\} \\
 & \qquad\qquad\qquad destroy(Self) \to Node\ )
\end{array}
$$

Fig. 8. TROLL *light* example in abstract syntax

The behavior part is graphically represented in Fig. 9. In order to refer to the current object which is to be defined we introduced the special constant $Self$. In the `BEHAVIOR` section, the concrete syntax of TROLL *light* is a little bit more general in allowing sequences of formulas and event terms separated by arrrows. However, by employing more process names, all TROLL *light* behavior patterns can be brought into the above form.

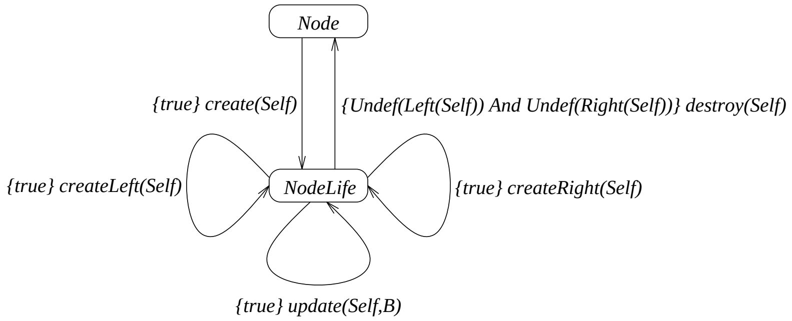

Fig. 9. Graphical representation of object behavior

In Fig. 10 we review the ingredients of the axioms and show their effect on object communities: Constraints and derivation rules have influence on A_L and A_R, interaction rules touch A_L and $\hat{A}$, and valuation rules and behavior definitions concern A_L, $\hat{A}$, and A_R. The above five kinds of axioms restricting object communities are special for TROLL *light* because at the beginning of the project we regarded these forms as a compromise between expresiveness and feasibility of implementation. Of course, many other choices are possible.

We now come to an important point with regard to the semantics of TROLL *light*, namely the distinction between attribute and subobject symbols which syntactically look the same. The main point is that subobject symbols are used to generate object identities. For a given collection of TROLL *light* templates, one template must be chosen as the schema template. Furthermore,

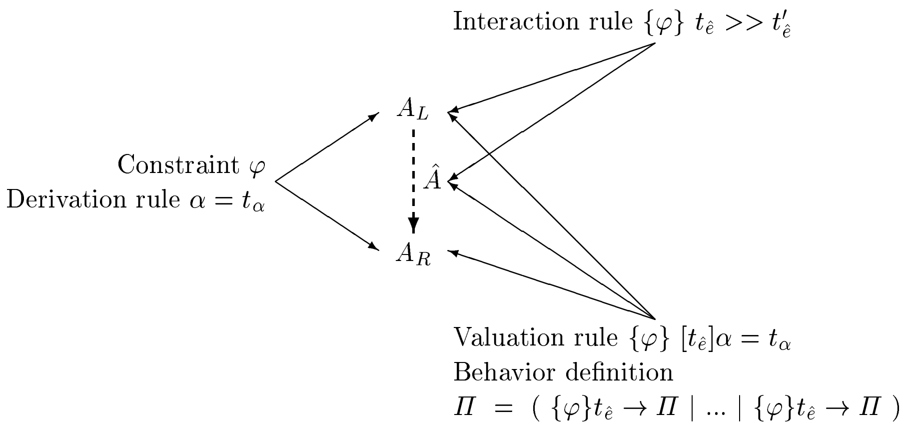

Fig. 10. Overview on axioms and their effect on object communities

there is one schema object belonging to the schema template which exists per se. All other objects are generated by finite birth event calling as direct or indirect subobjects of this schema object. This means that in each state only a finite number of objects can exist. Speaking in more technical terms, this is realized as follows.

Definition 21. Let a TROLL *light* specification and its induced object signature $O\Sigma$ and attribute signature $A\Sigma$ be given. The subset of the attribute symbols corresponding to the TROLL *light* subobject symbols is called the **subobject signature** $S\Sigma$.

Example 22 (Subobject signature). In our running example, the two symbols $left$ and $right$ constitute the subobject signature.

Definition 23. Let a subobject signature $S\Sigma$ and one distinguished schema template be given. The **object identities** are given by the term algebra generated by the subobject signature and by one distinguished constant i denoting the schema object and belonging to the object sort induced by the schema template: $T_{S\Sigma}(i)$. The notation i stands for "initially[5] existing object".

Example 24 (Object identities). In our running example we have the following object identities: $\{\ i,\ left(i),\ right(i),\ left(left(i)),\ right(left(i)),\ ...\ \}$.

Requirement 25 (Interpretation of object sorts and $\hat{e}$). The construction for object identities has severe consequences for TROLL *light* object communities. It restricts the interpretation for object sorts in the following way. If we have a triple $\langle A_L, \hat{A}, A_R\rangle$ in $M_{O\Sigma}$, then[6] $A_{L,os}, A_{R,os} \subseteq_{finite} T_{S\Sigma,os}(i)^{\perp}$ for $os \in OS$

[5] Initially is not meant in the category theory sense of the word.
[6] If X is a set, then $X^{\perp}$ stands for $X \cup \{\perp\}$.

and $\hat{A}_s \subseteq_{finite} T_{E\Sigma,s}(i)^{\perp}$ for $s \in OS \cup \{\hat{e}\}$. Thus, in each state there is only a finite number of objects, and each transition is caused by a finite number of events. Moreover, the interpretation of subobject symbols as attributes in A_L, $\hat{A}$, and A_R must coincide with the interpretation in $T_{S\Sigma,os}(i)$ resp. $T_{E\Sigma,s}(i)$ except for the special treatment of $\perp$. Additionally, all objects in A_L, $\hat{A}$, and A_R must be reachable from i via subobject connections. In consequence, the objects build a strict subobject hierarchy with i at its top. These requirements are already fulfilled, for instance, in the attribute algebra described in Example 6 and in the event algebra mentioned in Example 8.

Example 26 (Definedness of attribute `Count`*).* We now can explain why the attribute `Count` in Example 14 is well-defined: The `Node` objects in our running example build a strict hierarchy. Therefore, every `Count` computation terminates after a finite number of recursively triggered `Count` computations.

Requirement 27 (One event per object). In order to make things easier for the implementation of the TROLL *light* animator [HG94b] we have required that at most one event can occur in one object during one transition. But this is not a general requirement one has to make.

Requirement 28 (Frame rule). In the semantics of TROLL *light* we have also an implicit frame rule concerning the valuation rules: If we have a transition $\langle A_L, \hat{A}, A_R \rangle$ in $M_{O\Sigma}$, then changes of attribute values are implied by valuation rules triggered by events occurring in $\hat{A}$, and there are no other changes of attribute values. More formally, we require for $\langle A_L, \hat{A}, A_R \rangle$ in $M_{O\Sigma}$ the following to be true[7].

$$[\![\alpha(x)]\!]_{A_R} = \begin{cases} [\![t_\alpha]\!]_{A_L} & \text{if there is a valuation rule } \{\varphi\}[t_{\hat{e}}]\alpha = t_\alpha \text{ such} \\ & \text{that } \varphi \text{ holds in } A_L \text{ and } t_{\hat{e}} \text{ occurs in } \hat{A} \\ [\![\alpha(x)]\!]_{A_L} & \text{otherwise} \end{cases}$$

TROLL *light* does not have a loose semantics in the sense that a specification is associated with a class of models, but the semantics of a TROLL *light* text is one distinguished object community.

Definition 29. Let an object signature $O\Sigma$, and sets of constraints, valuation, derivation, interaction rules, and behavior definitions, i.e., a TROLL *light* specification, be given. The **specified object community** is the minimal[8] object community which (1) satisfies the given axioms, (2) obeys the Requirements 25, 27, and 28, and (3) is generated by (3.a) starting in an empty attribute algebra and performing a birth event for the schema object i and by (3.b) successively generating all other attribute algebras by performing all possible finite sets of events.

[7] The notation $[\![t]\!]_A$ stands for the value of t in the algebra A.

[8] Minimal is meant in the sense of set theory. We have worked out a notion of object community morphism, and our semantics for TROLL *light* is an initial semantics in an appropriate category. These details, however, are beyond the scope of this paper.

In general, the model class associated with a TROLL *light* specification may be empty. This is because we allow (some variant of) first-order predicate logic to be used for constraints, pre-conditions in behavior patterns, etc. Therefore it is possible to specify contradictions.

Example 30 (Semantics of the running TROLL light example). The previously explained object community in Example 10 was already the object community which we associate with our running example.

4 Relation to Other Approaches

Due to space limitations, a detailed comparison between our and other approaches is beyond the scope of this paper. We can only briefly comment on existing work. Our proposal is along the informal line presented in [EO94]. There, a four-level proposal for dynamic abstract data type is presented. Their level one (value type specification) corresponds to our data signatures and data algebras, their level two (instant structure specification) to our attribute signatures and attribute algebras, and their level three (dynamic operations specification) to our event signatures and event algebras. Their level four (higher level dynamic specification) would correspond to parametrized object signatures and specifications.

Another recent proposal combining algebraic specifications and object-oriented ideas can be found in [AZ94]. But, in contrast to [AZ94] we do not have events returning values. Instead, we have the possiblity of querying the system states in an SQL-like style [HG94a]. A predecessor of [AZ94] was the entity algebra approach introduced in [Reg91].

Algebraic specifications with implicit state are proposed in [DG94]. Their extension of PLUSS looks very close to TROLL *light*. There are also proposals around to embed object-oriented ideas into conventional algebraic specifications, for instance [GM87, Bre91, GD94]. Another line of research are extensions of the Berlin module approach [EW85] into the direction of object-oriented programming [GR91, PPP94] concentrating on the relationship between algebraic data types and classes in the sense of object-oriented programming. In [Rei94] a semantics for objects is given in terms of pure category theory. Close to our approach are also the transformation rule approach [BHK84] and the evolving algebras [Gur88] and their applications [GKS91]. Last but not least, within the IS-CORE working group a plenty of logic-based approaches like [FM92, SSC92, EDS93] are around. In contrast to them, we here put emphasis on pure model-theoretic aspects.

5 Conclusion and Future Work

We shortly introduced the specification language TROLL *light* and mentioned its development environment. We have pointed out a standard semantics for

TROLL *light* in pure algebraic terms obtaining a relative simple model for an object community.

We hope, this simplicity may open research directions for other important issues like parametrization, modularization, or implementation of object communities. In particular, we want to study the relation of our approach to the level four (higher level dynamic specification) of the informal proposal made in [EO94]. Another exciting question is what other types of axioms are helpful in specifying structure and behavior of object communities.

Acknowledgements

The authors have benefitted from comments of the referees and various COMPASS people. TROLL *light* was developed in cooperation with (former) Braunschweig collegues.

References

[AAR94] R. Achuthan, V.S. Alagar, and T. Radhakrishnan. An Object-Oriented Framework for Specifying Reactive Systems. In V.S. Alagar and R. Missaoui, editors, *Proc. Colloquium on Object Orientation in Databases and Software Engineering (COODBSE'94)*, pages 18–30. Université du Quebéc à Montréal, 1994.

[AZ94] E. Astesiano and E. Zucca. D-Oids: A Model for Dynamic Data-Types. *Mathematical Structures in Computer Science*, 1994.

[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 Opérationnelle, Publication 748, Université de Montréal, 1990.

[BHK84] J.A. Bergstra, J. Heering, and J.W. Klop. Object-Oriented Algebraic Specification: Proposal for a Notation and 12 Examples. Technical Report CS-R8411, CWI, Department of Computer Science, Amsterdam, 1984.

[Bre91] R. Breu. *Algebraic Specification Techniques in Object Oriented Programming Environments*. Springer, Berlin, LNCS 562, 1991.

[CGH92] S. Conrad, M. Gogolla, and R. Herzig. TROLL *light*: A Core Language for Specifying Objects. Informatik-Bericht 92–02, TU Braunschweig, 1992.

[CJO94] S. Clerici, R. Jimenez, and F. Orejas. Semantic Constructions in the Specification Language GLIDER. In H. Ehrig and F. Orejas, editors, *Recent Trends in Data Type Specification (WADT'92)*, pages 144–157. Springer, Berlin, LNCS 785, 1994.

[DG94] P. Dauchy and M.-C. Gaudel. Algebraic Specifications with Implicit State. Technical Report 887, Université de Paris-Sud, 1994.

[DO91] O.-J. Dahl and O. Owe. Formal Development with ABEL. Technical Report 159, University of Oslo, 1991.

[EDS93] H.-D. Ehrich, G. Denker, and A. Sernadas. Constructing Systems as Object Communities. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proc. Theory and Practice of Software Development (TAPSOFT'93)*, pages 453–467. Springer, Berlin, LNCS 668, 1993.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics.* Springer, Berlin, 1985.

[EM90] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 2: Modules and Constraints.* Springer, Berlin, 1990.

[EO94] H. Ehrig and F. Orejas. Dynamic Abstract Data Types: An Informal Proposal. *EATCS Bulletin*, 53:162–169, 1994.

[EW85] H. Ehrig and H. Weber. Algebraic Specification of Modules. In E.J. Neuhold and G. Chroust, editors, *Formal Models in Programming – Proc. of the IFIP TC2 Working Conf. on the Role of Abstract Models in Information Processing*, pages 231–258. North-Holland, Amsterdam, 1985.

[FM92] J. Fiadeiro and T. Maibaum. Temporal Theories as Modularisation Units for Concurrent System Specification. *Formal Aspects of Computing*, 4(3):239–272, 1992.

[Gab93] P. Gabriel. The Object-Based Specification Language Π: Concepts, Syntax, and Semantics. In M. Bidoit and C. Choppy, editors, *Recent Trends in Data Type Specification (WADT'91)*, pages 254–270. Springer, LNCS 655, 1993.

[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'93)*, pages 17–32. Springer, Berlin, LNCS 722, 1993.

[GD94] J.A. Goguen and R. Diaconescu. Towards an Algebraic Semantics for the Object Paradigm. In H. Ehrig and F. Orejas, editors, *Recent Trends in Data Type Specification (WADT'92)*, pages 1–29. Springer, LNCS 785, 1994.

[GHC+94] M. Gogolla, R. Herzig, S. Conrad, G. Denker, and N. Vlachantonis. Integrating the ER Approach in an OO Environment. In R. Elmasri, V. Kouramajian, and B. Thalheim, editors, *Proc. 12th Int. Conf. on the Entity-Relationship Approach (ER'93)*, pages 376–389. Springer, Berlin, LNCS 823, 1994.

[GKS91] G. Gottlob, G. Kappel, and M. Schrefl. Semantics of Object-Oriented Data Models – The Evolving Algebra Approach. In J.W. Schmidt and A.A. Stogny, editors, *Proc. 1st Int. East-West Database Workshop*, pages 144–160. Springer, Berlin, LNCS 504, 1991.

[GM87] J.A. Goguen and J. Meseguer. Unifying Functional, Object-Oriented and Relational Programming with Logical Semantics. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 417–477. MIT Press, 1987.

[Gog89] M. Gogolla. Algebraization and Integrity Constraints for an Extended Entity-Relationship Approach. In J. Diaz and F. Orejas, editors, *Proc. Theory and Practice of Software Development (TAPSOFT'89)*, pages 259–274. Springer, Berlin, LNCS 351, 1989.

[GR91] M. Grosse-Rhode. Towards Object-Oriented Algebraic Specifications. In H. Ehrig, K.P. Jantke, F. Orejas, and H. Reichel, editors, *Recent Trends in Data Type Specification (WADT'90)*, pages 98–116. Springer, Berlin, LNCS 534, 1991.

[Gur88] Y. Gurevich. Logic and the Challenge of Computer Science. In E. Börger, editor, *Trends in Theoretical Computer Science*, pages 1–57. Computer Science Press, Rockville (MD), 1988.

[GVH+94] M. Gogolla, N. Vlachantonis, R. Herzig, G. Denker, S. Conrad, and H.-D. Ehrich. The KORSO Approach to the Development of Reliable Information Systems. Technical Report 94-06, TU Braunschweig, 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)*, pages 183–194. Cépaduès-Éditions, Toulouse, 1994.

[Hen88] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, Cambridge (MA), 1988.

[HG94a] R. Herzig and M. Gogolla. A SQL-like Query Calculus for Object-Oriented Database Systems. In E. Bertino and S. Urban, editors, *Proc. Int. Symp. on Object-Oriented Methodologies and Systems (ISOOMS'94)*, pages 20–39. Springer, Berlin, LNCS 858, 1994.

[HG94b] R. Herzig and M. Gogolla. An Animator for the Object Specification Language TROLL *light*. In V.S. Alagar and R. Missaoui, editors, *Proc. Colloquium on Object Orientation in Databases and Software Engineering (COODBSE'94)*, pages 4–17. Université du Quebéc à Montréal, 1994.

[HK87] R. Hull and R. King. Semantic Database Modelling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs (NJ), 1985.

[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.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, Englewood Cliffs (NJ), 1989.

[PM88] J. Peckham and F. Maryanski. Semantic Data Models. *ACM Computing Surveys*, 20(3):153–189, 1988.

[PPP94] F. Parisi-Presicce and A. Pierantonio. Structured Inheritance for Algebraic Class Specifications. In H. Ehrig and F. Orejas, editors, *Recent Trends in Data Type Specification (WADT'92)*, pages 295–309. Springer, Berlin, LNCS 785, 1994.

[Reg91] G. Reggio. Entities: An Institution for Dynamic Systems. In H. Ehrig, K.P. Jantke, F. Orejas, and H. Reichel, editors, *Recent Trends in Data Type Specification (WADT'90)*, pages 246–265. Springer, LNCS 534, 1991.

[Rei94] H. Reichel. An Approach to Object Semantics Based on Terminal Co-Algebras. *Mathematical Structures in Computer Science*, 1994.

[SSC92] A. Sernadas, C. Sernadas, and J.F. Costa. Object Specification Logic. Internal Report, INESC, University of Lisbon, 1992. To appear in Journal of Logic and Computation.

[SSE87] A. Sernadas, C. Sernadas, and H.-D. Ehrich. Object-Oriented Specification of Databases: An Algebraic Approach. In P.M. Stocker and W. Kent, editors, *Proc. 13th Int. Conf. on Very Large Data Bases (VLDB'87)*, pages 107–116. Morgan-Kaufmann, Palo Alto, 1987.

[Wie91] R. 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 WG 2.6 Working Conference, Windermere (UK) 1990*, pages 415–438. North-Holland, 1991.

[Wir90] M. Wirsing. Algebraic Specification. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 677–788. North-Holland, Amsterdam, 1990.

This article was processed using the LaTeX macro package with LLNCS style