Parameterizing Object Specifications

Martin Gogolla

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

Abstract. We present a proposal for parameterized object specifications allowing especially objects sorts in the parameter. These object specifications permit to describe the part of the world to be modeled as an object community of concurrently existing and communicating objects. Our proposal for parameter passing works well on the syntactical level by means of a pushout construction. On the semantic level we use free constructions and corresponding forgetful functors.

1 Motivation

In the software development process usually two important phases are distinguished, the requirements engineering and the design engineering phase. At the end of the requirements engineering phase, a first formal description of the system in mind should be the "contract" for further development. Therefore, the task of delivering this formal specification is of central concern.
Formal specification calls for suitable specification techniques. It is an open debate whether there exists a general description technique suitable for any system development, or whether special systems call for special specification techniques with made-to-order expressive power.
In order to support the specification of reliable information systems in an adequate way, we developed a simple language for the description of structural and dynamic properties of information system objects, called TROLL *light* [CGH92, GCH93, HCG94]. 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.
TROLL *light*, a dialect of OBLOG [SSE87] and TROLL [JHSS91], incorporates ideas from algebraic specification [EM85, EM90, Wir90], semantic data models [HK87, PM88], and process theory [Hoa85, Hen88, Mil89]. Object properties are formalized in the context of so-called templates (or object types) explained here by means of `Author` objects.

```
TEMPLATE Author
  ...
  ATTRIBUTES
    Name:string; DateOfBirth:date;
    Books:LIST(book);
```

```
  ...
  EVENTS
    BIRTH create(Name:string, DateOfBirth:date);
    publishBook(Title:string,Publisher:string);
  ...
END TEMPLATE;
```

In the database context objects of the same type are usually collected in certain containers called classes[1]. It is a distinguishing feature of TROLL *light* that the basic template concept is also used for the description of those classes, namely in the form of composite objects.

```
TEMPLATE AuthorContainer
  ...
  SUBOBJECTS
    Authors(No:nat):author;
  ...
  EVENTS
    ...
    addAuthor(No:nat);
    removeAuthor(No:nat);
    ...
END TEMPLATE;
```

Among other things, the given specification tells us that in the context of an object of type `AuthorContainer` authors are distinguished by natural numbers. Hence `No:nat` serves as a key.

It is clear that in the description of large object communities many containers of the given kind will arise which mainly differ only in the type of the contained objects and the used keys. This directly leads to the idea of parameterized object specifications. The aim of this paper is to find a formal foundation for such specifications.

The structure of the rest of the paper is as follows. In Sect. 2 we briefly explain non-parameterized object specifications examined in detail in [GH95]. In Sect. 3 we consider parameterization, a well-investigated technique in the area of algebraic specification. The main results are on the syntactical side a pushout construction for the application of parameterized specifications to actual ones and a combination of free and forgetful functor constructions for the semantic part. Section 4 puts our approach into the context of other recent research.

In Sect. 5 we give some concluding remarks and mention future work to be done. Due to space limitations we cannot go into details of a comparision of our approach to other recent research [GM87, GR91, Reg91, FM92, SSC92, EDS93, AZ94, DG94, EO94, GD94, PPP94] but must refer for this topic to [GH95].

[1] This notion of class is different from the notion of class in object-oriented programming where object types are called classes.

2 Non-Parameterized Object Specifications

Non-parameterized object specifications were examined in detail in [GH95]. The basic idea consists in viewing object states as attribute algebras and transitions between attribute algebras as so-called event algebras. This section reviews the main ideas.

2.1 Syntax

We first consider the syntax of (TROLL *light*) object specifications. Objects are described on the basis of conventional data type specifications. The basic difference between data values as instances of data types and objects as instances of object types is that values are regarded as stateless items while objects are associated with an inherent notion of state.
Technically speaking object signatures are built of a data signature part (consisting of data sorts and corresponding data operators) and object specific items like object (identifier) sorts, attribute symbols and event symbols.

Definition 1. An **object signature** $O\Sigma = (DS, \Omega, OS, A, E)$ consists of

- a set DS of data sorts,
- a family of sets of operation symbols $\Omega = \langle \Omega_{dw,ds} \rangle_{dw \in DS^*, ds \in DS}$,
- a set OS of object sorts,
- 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 data and object sorts: $S = DS \cup OS$.

Example 2. The template `Author` induces the following object signature.

$$
\begin{aligned}
DS &= \{\ string, date\} \\
\Omega &= \{\ concat : string \times string \rightarrow string, \\
&\qquad day : date \rightarrow nat, \ldots\} \\
OS &= \{\ author, book\} \\
A &= \{\ Name : author \rightarrow string, \\
&\qquad DateOfBirth : author \rightarrow date, \ldots\} \\
E &= \{\ create : author \times string \times date, \\
&\qquad publishBook : author \times string \times string, \ldots\}
\end{aligned}
$$

In the language TROLL *light* every template induces a corresponding object identifier sort. The relation is demonstrated by the convention that the same names are used for templates and corresponding object identifier sorts with the exception that a template name begins with an upper case letter while the corresponding object identifier sort starts with a lower case letter. With respect to subobject symbols as found in `AuthorContainer` we assume that these are treated just like attribute symbols. The subtle distinction is discussed in [GH95].
The interpretation of data sorts, data operations, object identifier sorts, and attribute symbols characterize the states of an object community. Event symbols are associated with state transitions. Hence we define the notions of data, attribute, and event signatures as follows.

Definition 3. Let an object signature $O\Sigma$ be given. The **data signature** induced by $O\Sigma$ consists of the data sorts and the data operations of the object signature, i.e., $D\Sigma = (DS, \Omega)$.
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., $A\Sigma = (DS \cup OS, \Omega \cup A)$.
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 $\hat{E}_{os,w,\hat{e}} := E_{os,w}$.

The above signatures are signatures in the classical sense. Thus, they induce corresponding classes of algebras. We will use the notions data, attribute, and event algebra to refer to them.
Besides the signature part TROLL *light* templates further include certain axioms restricting the interpretation of object signatures.

Definition 4. Let an object signature $O\Sigma$ be given. The **axioms** for object community specification are divided into constraints, derivation rules, valuation rules, interaction rules, and behavior definitions.

- A **constraint** φ is a formula restricting the class of possible attribute algebras $ALG_{A\Sigma}$.
- 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 **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.
- An **interaction rule** is of the form $\{\varphi\}\ t_{\hat{e}} \gg t'_{\hat{e}}$ where φ is a formula, and $t_{\hat{e}}$ and $t'_{\hat{e}}$ are event terms.
- A **behavior definition** for a template T consists of a set of process definitions of the following form:
  $\Pi_0 = (\{\varphi_1\}t_{\hat{e}_1} \rightarrow \Pi_1 \mid \{\varphi_2\}t_{\hat{e}_2} \rightarrow \Pi_2 \mid \ldots \mid \{\varphi_n\}t_{\hat{e}_n} \rightarrow \Pi_n)$
  Here the Π_i's are process names, the φ_i's formulas, and the $t_{\hat{e}_i}$'s event terms.

Informally speaking, valuation rules are used to express the effect of events on attributes, interaction rules allow to synchronize events in different objects thereby supporting communication between objects, and with behavior definitions allowed event sequences can be specified.

2.2 Semantics

Object signatures are interpreted by so-called object communities. This name was chosen because in object communities different objects co-exist and communicate with each other.

Definition 5. Let an object signature $O\Sigma$ be given. $MOD_{O\Sigma}$ denotes the **class** of all **$O\Sigma$-models or $O\Sigma$-object communities**:

$$MOD_{O\Sigma} = \{M_{O\Sigma} \mid M_{O\Sigma} \subseteq (ALG_{A\Sigma} \times ALG_{E\Sigma} \times ALG_{A\Sigma})\}$$

We may assume the interpretation of the data signature to be fixed. It is also possible to restrict the interpretation of object identifier sorts in the sense that object identifiers will be chosen from a term algebra induced by the attribute signature. Then, only attributes contribute to states. For details we have to refer to [GH95] again.
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 occurence of the events in $\hat{A}$. A step in an object's life can be traced along such triples: For example, an object $obj \in A_{L,os}$ for some $os \in OS$ can have for attribute α the value c_1 in A_L $[\alpha_{A_L}(obj, ...) = c_1]$; this can change in A_R to c_2 $[\alpha_{A_R}(obj, ...) = c_2]$ due to the occurrence of event e in $\hat{A}$ $[e_{\hat{A}}(obj, ...) \neq \bot]$. 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 $A\Sigma$.
The possible state transition relation must obey the specified axioms. Therefore one defines the validity of axioms as follows.

Definition 6. Let an object signature $O\Sigma$, an $O\Sigma$-model $M_{O\Sigma}$ and axioms over $O\Sigma$ be given. The **validity** of axioms is defined as follows.

- A constraint φ is **valid** in $M_{O\Sigma}$ iff for all triples $\langle A_L, \hat{A}, A_R \rangle$ in $M_{O\Sigma}$ the formula φ holds in A_L and A_R.
- A derivation rule $\alpha = t_\alpha$ 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.
- A valuation rule $\{\varphi\}[t_{\hat{e}}]\alpha = t_\alpha$ 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[2] in $\hat{A}$, then the attribute α evaluates in A_R to the value which t_α had in A_L.
- An interaction rule $\{\varphi\}$ $t_{\hat{e}} \gg t'_{\hat{e}}$ 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 $\hat{A}$.
- A behavior definition
  $\Pi_0 = (\{\varphi_1\}t_{\hat{e}_1} \to \Pi_1 \mid \{\varphi_2\}t_{\hat{e}_2} \to \Pi_2 \mid ... \mid \{\varphi_n\}t_{\hat{e}_n} \to \Pi_n)$
  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 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, $process\_state(o) = \Pi_0$ in A_L, and $process\_state(o) = \Pi_j$ in A_R. In order to handle behavior definitions in a correct way, the object signature $O\Sigma$ has to be extended by attributes $process\_state : os \to process\_state\_sort$ for every object sort $os \in OS$.

Definition 7. Let an object signature $O\Sigma$ and a set AX of axioms over $O\Sigma$ be given. The **class** of all **$O\Sigma$-models** satisfying the **axioms** AX is denoted by $MOD_{O\Sigma,AX}$.

In order to explain later in Def. 19 the semantics of parameterized object specifications we need a notion of morphism between object communities.

Definition 8. Let an object signature $O\Sigma$ and object communities $M, M' \in MOD_{O\Sigma}$ be given. A triple $\langle f_L, \hat{f}, A_R \rangle$ of mappings $f_L : M_L \to M'_L$, $\hat{f} : \hat{M} \to \hat{M}'$, and

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

$f_R : M_R \to M'_R$ is called an **object community morphism** iff $\langle A_L, \hat{A}, A_R\rangle \in M$ implies $\langle f_L(A_L), \hat{f}(\hat{A}), f_R(A_R)\rangle \in M'$. Here M_L denotes the restriction of M to the first component, i.e., $M_L := \{A_L \mid \langle A_L, \hat{A}, A_R\rangle \in M\}$. $\hat{M}$ and M_R are defined analogously.

3 Parameterized Object Specifications

One of the main motivation for considering parameterized object specifications is the possibility of reusing specifications. A parameterized object specification consists of a formal parameter part and a body part. A parameterized specification can be applied to different actual parameters yielding different versions of the body specification. Another main advantage of employing parameterized specifications is modularity. This means that one can split large specifications into smaller manageable pieces.

3.1 Syntax

In order to compare different specifications, for instance parameter specification and body specification or parameter specification and actual specification, specifications must become in some sense comparable. The technical tool for this are signature morphisms.

Definition 9. Let object signatures $O\Sigma_1$ and $O\Sigma_2$ be given. A **signature morphism** $f : O\Sigma_1 \to O\Sigma_2$ is given by mappings

- $f_{DS} : DS_1 \to DS_2$,
- $f_\Omega : \Omega_1 \to \Omega_2$ compatible with f_{DS},
- $f_{OS} : OS_1 \to OS_2$,
- $f_A : A_1 \to A_2$ compatible with f_{DS} and f_{OS}, and
- $f_E : E_1 \to E_2$ compatible with f_{DS} and f_{OS}.

Fact 10. The category with object signatures as (category) objects and signature morphism as (category) morphisms has pushouts.

Definition 11. A **parameterized signature** is a pair of signatures $PO\Sigma = (PDS, P\Omega, POS, PA, PE) \subseteq BO\Sigma = (BDS, B\Omega, BOS, BA, BE)$ where the parameter signature $PO\Sigma$ is included in the body signature $BO\Sigma$.

Definition 12. A **parameterized object specification** $(PO\Sigma, PAX) \subseteq (BO\Sigma, BAX)$ is given by a parameterized signature and parameter axioms PAX and body axioms BAX.

Example 13 (Container). The first example for a parameterized object specification introduces a container which holds objects belonging to the following parameter signature.

$$\begin{aligned}
DS &= \{data_sort\}\\
\Omega &= \emptyset\\
OS &= \{entry_sort\}\\
A &= \emptyset\\
E &= \{create_entry : entry_sort, destroy_entry : entry_sort\}
\end{aligned}$$

The body signature gives an object sort for container objects and provides events for insertion and deletion.

$$\begin{array}{ll} DS & = \emptyset \\ \Omega & = \emptyset \\ OS & = \{container\_sort\} \\ A & = \{entries : container\_sort \times data\_sort \rightarrow entry\_sort\} \\ E & = \{\, create\_container : container\_sort, \\ & \quad destroy\_container : container\_sort, \\ & \quad add\_entry : container\_sort \times data\_sort, \\ & \quad remove\_entry : container\_sort \times data\_sort\} \end{array}$$

It should be clear that the respectives body parts show only the differences from the parameter parts. Thus for instance, in the body we really have $OS = \{entry\_sort, container\_sort\}$. The body axioms synchronize the insertion and deletion events in the container with the given parameter events which are assumed to be birth and death events for entry objects.

$$add\_entry(self, x) \gg create\_entry(entries(self, x))$$
$$remove\_entry(self, x) \gg destroy\_entry(entries(self, x))$$

The *data_sort* of the parameter serves in the body as a kind of key attribute (in the sense of semantic data models) because entry objects are identified by its container c and a data value v belonging to the *data_sort*: $entries(c, v)$ gives an entry object.

Example 14 (Synchronization). Quite another example for a parameterized object specification is the following one which mainly serves to synchronize two events. The parameter signature gives these two events.

$$\begin{array}{ll} DS & = \emptyset \\ \Omega & = \emptyset \\ OS & = \{obj\_sort_1, obj\_sort_2\} \\ A & = \emptyset \\ E & = \{event_1 : obj\_sort_1, event_2 : obj\_sort_2\} \end{array}$$

The body signature adds a new object sort *synchronizer* which is intended to serve as a common super-object.

$$\begin{array}{ll} DS & = \emptyset \\ \Omega & = \emptyset \\ OS & = \{synchronizer\} \\ A & = \{\, obj_1 : synchronizer \rightarrow obj\_sort_1, \\ & \quad obj_2 : synchronizer \rightarrow obj\_sort_2\} \\ E & = \{create\_synchronizer : synchronizer \times obj\_sort_1 \times obj\_sort_2\} \end{array}$$

The body axioms simply require that $event_1$ occurs in the first object if and only if $event_2$ in the second object occurs.

$$[create\_synchronizer(self, x_1, x_2)]\ obj_1(self) = x_1, obj_2(self) = x_2$$
$$event_1(obj_1(self)) \gg event_2(obj_2(self))$$
$$event_2(obj_2(self)) \gg event_1(obj_1(self))$$

Example 15 (Specialization). An example taken from the area of semantic data models is the specialization specification presented hereafter. The parameter signature provides an object sort, an attribute, and an event for this sort.

$$
\begin{aligned}
DS &= \{data\_sort_1, data\_sort_2\} \\
\Omega &= \emptyset \\
OS &= \{original\_sort\} \\
A &= \{original\_attribute : original\_sort \rightarrow data\_sort_1\} \\
E &= \{original\_event : original\_sort\}
\end{aligned}
$$

The body signature introduces an object-valued attribute, which will be pointing to the object to be specialized, the specialized attribute, the specialized event, a new attribute, and a new event.

$$
\begin{aligned}
DS &= \emptyset \\
\Omega &= \emptyset \\
OS &= \{specialized\_sort\} \\
A &= \{\, who\_am\_i : specialized\_sort \rightarrow original\_sort, \\
&\qquad specialized\_attribute : specialized\_sort \rightarrow data\_sort_1, \\
&\qquad new\_attribute : specialized\_sort \rightarrow data\_sort_2\} \\
E &= \{\, create\_specialization : specialized\_sort \times original\_sort, \\
&\qquad specialized\_event : specialized\_sort, \\
&\qquad new\_event : specialized\_sort \times data\_sort_2\}
\end{aligned}
$$

The body axioms serve to inherit the original attribute and the original event into the specialized object. The main technical tool here is a derivation rule for attribute inheritance and an event calling rule for event inheritance.

$$
\begin{aligned}
&[create\_specialization(self, x)]\ who\_am\_i(self) = x \\
&[new\_event(self, y)]\ new\_attribute(self) = y \\
&specialized\_attribute(self) = original\_attribute(who\_am\_i(self)) \\
&specialized\_event(self) \gg original\_event(who\_am\_i(self))
\end{aligned}
$$

Let us emphasize one point here: What the parameterized object specification above achieves is more or less a formal specification of an inheritance mechanism. Thus it may be regarded as as the formal semantics of this inheritance concept. Of course, specifications like the above one are not assumed to be written down in a comfortable specification language but more a means to express semantics of suitable language features.

We now come to the application of parameterized object specifications to actual ones. This is done in the spirit of the classical pushout approach of [EKT+84].

Definition 16. Let a parameterized object specification $(PO\Sigma, PAX) \subseteq (BO\Sigma, BAX)$ and an actual object specification $(AO\Sigma, AAX)$ together with a signature morphism $f : PO\Sigma \rightarrow AO\Sigma$ be given. The result of the **application** of $(PO\Sigma, PAX) \subseteq (BO\Sigma, BAX)$ to $(AO\Sigma, AAX)$ via the parameter passing morphism $f : PO\Sigma \rightarrow AO\Sigma$ is the pushout object specification $(RO\Sigma, RAX)$ of the following diagram.

$$
\begin{array}{ccc}
(PO\Sigma, PAX) & \xhookrightarrow{i} & (BO\Sigma, BAX) \\
f \big\downarrow & po & f' \big\downarrow \\
(AO\Sigma, AAX) & \xrightarrow{i'} & (RO\Sigma, RAX)
\end{array}
$$

Example 17 (Application of a parameterized object specification). Suppose that we have appropriate non-parameterized object specifications for authors and books.

Then we can apply the parameterized object specification container to the author specification by the signature morphism f:

$$\begin{aligned} f: & \ data\_sort && \mapsto nat \\ & \ entry\_sort && \mapsto author \\ & \ create\_entry && \mapsto create\_author \\ & \ destroy\_entry && \mapsto destroy\_author \end{aligned}$$

And we can also apply the container specification to the book specification by the signature morphism f':

$$\begin{aligned} f': & \ data\_sort && \mapsto string \\ & \ entry\_sort && \mapsto book \\ & \ create\_entry && \mapsto create\_book \\ & \ destroy\_entry && \mapsto destroy\_book \end{aligned}$$

Thus in the respective containers, author objects are identified by natural numbers and book objects are identified by strings.

3.2 Semantics

Fact 18. A signature morphism $f : O\Sigma_1 \to O\Sigma_2$ induces a corresponding translation $\mathcal{U}_f : MOD_{O\Sigma_2} \to MOD_{O\Sigma_1}$ on models by defining for $M_{O\Sigma_2} \in MOD_{O\Sigma_2}$

$$\begin{aligned} \mathcal{U}_f(M_{O\Sigma_2}) := \{ & \langle \mathcal{U}_{A\Sigma_2 \to A\Sigma_1}(A_L), \mathcal{U}_{E\Sigma_2 \to E\Sigma_1}(\hat{A}), \mathcal{U}_{A\Sigma_2 \to A\Sigma_1}(A_R) \rangle \mid \\ & \langle A_L, \hat{A}, A_R \rangle \in M_{O\Sigma_2} \} \end{aligned}$$

Definition 19. The **semantics of a parameterized object specification** is given by a function $\mathcal{F} : MOD_{PO\Sigma,PAX} \to MOD_{BO\Sigma,BAX}$ defined as follows.

- Because the parameter attribute signature is included in body attribute signature $PA\Sigma \hookrightarrow BA\Sigma$ and the parameter event signature is included in body event signature $PE\Sigma \hookrightarrow BE\Sigma$, we have free constructions from parameter attribute algebras to body attribute algebras and from parameter event algebras to body event algebras.

  $$\begin{aligned} \mathcal{F}_{PA\Sigma \to BA\Sigma} : ALG_{PA\Sigma} \to ALG_{BA\Sigma} \\ \mathcal{F}_{PE\Sigma \to BE\Sigma} : ALG_{PE\Sigma} \to ALG_{BE\Sigma} \end{aligned}$$
- But in general, the carrier sets of object sorts appearing only in the parameter will be empty due to the freeness of the construction. Therefore, we have to add means to generate object identities. We do this by enlarging the body signature by special operations for each object sort os appearing in the body but not in the parameter:

  $$\begin{aligned} first_{os} & : \to os \\ last_{os} & : os \to os \end{aligned}$$

  The obtained signature is called $BA\Sigma'$. Now we have a free construction $\mathcal{F}_{PA\Sigma \to BA\Sigma'}$ from parameter attribute algebras to enlarged body algebras. The opposite to the free construction $\mathcal{F}_{BA\Sigma \to BA\Sigma'}$ from body algebras to enlarged body algebras is given by the respective forgetful functor $\mathcal{U}_{BA\Sigma' \to BA\Sigma}$ from enlarged body algebras to body algebras. Analogous constructions work for event algebras. In the total we get the following constructions.

  $$\begin{aligned} \mathcal{F}_{PA\Sigma} := \mathcal{F}_{PA\Sigma \to BA\Sigma'} \circ \mathcal{U}_{BA\Sigma' \to BA\Sigma} \\ \mathcal{F}_{PE\Sigma} := \mathcal{F}_{PE\Sigma \to PA\Sigma'} \circ \mathcal{U}_{PA\Sigma' \to PA\Sigma} \end{aligned}$$

- For a given $(PO\Sigma, PAX)$-model $M_{PO\Sigma,PAX}$ we can therefore define the following classes of possible $BA\Sigma$- and $BE\Sigma$-algebras.
$$POS_{BA\Sigma} := \{ \mathcal{F}_{PA\Sigma}(A_L) \mid \langle A_L, \hat{A}, A_R \rangle \in M_{PO\Sigma,PAX} \} \cup \{ \mathcal{F}_{PA\Sigma}(A_R) \mid \langle A_L, \hat{A}, A_R \rangle \in M_{PO\Sigma,PAX} \}$$
$$POS_{BE\Sigma} := \{ \mathcal{F}_{PE\Sigma}(\hat{A}) \mid \langle A_L, \hat{A}, A_R \rangle \in M_{PO\Sigma,PAX} \}$$
- Employing these classes we can define a class of possible body models satisfying the axioms as follows.
$$POS_{BO\Sigma,BAX} := \{ M \mid M \subseteq (POS_{BA\Sigma} \times POS_{BE\Sigma} \times POS_{BA\Sigma}) \wedge M \models BAX \}$$
- The semantics of the complete construction is the least model in this set (if it exists)
$$\mathcal{F}(M_{PO\Sigma,PAX}) := M_{least}$$
such that there exists an object community morphism $f : M \to M_{least}$ for all $M \in POS_{BO\Sigma,BAX}$.

Remark 20. In general, the function $\mathcal{F}$ will not be persistent in the sense that a given parameter model $M_{PO\Sigma,PAX}$ will resist in $\mathcal{F}(M_{PO\Sigma,PAX})$. Speaking in more technical terms, $M_{PO\Sigma,PAX}$ and $\mathcal{U}_i(\mathcal{F}(M_{PO\Sigma,PAX}))$ will in general be not isomorphic ($\mathcal{U}_i$ refers to the translation of $BO\Sigma$-models to $PO\Sigma$-models). This is demonstrated by Example 14 for synchronization. There the parameter object society will be restricted in its behavior by the additional synchronization rule, and possible life cycles of the parameter object society are forbidden in the resulting body object society. Another property of $\mathcal{F}$ is partiality. It is easy to write inconsistent object specifications due to the existence of constraints restricting the possible attribute algebras. In the extreme case one could require $x \neq x$ as a constraint resulting in an empty model class for the object specification. If such a constraint would be part of the body axioms one would get for $\mathcal{F}$ the completely undefined function. On the other hand, one can also imagine situations where $\mathcal{F}$ is defined for a model $M_{PO\Sigma,PAX}$ but undefined for $M'_{PO\Sigma,PAX}$ (perhaps due to some constraint which holds in $\mathcal{F}(M_{PO\Sigma,PAX})$ but not in $\mathcal{F}(M'_{PO\Sigma,PAX})$).

Definition 21. Let a parameterized object specification $(PO\Sigma, PAX) \subseteq (BO\Sigma, BAX)$ and an actual object specification $(AO\Sigma, AAX)$ together with a signature morphism $f : PO\Sigma \to AO\Sigma$ be given. Parameter passing is called **correct** if $\mathcal{U}_f(MOD_{AO\Sigma,AAX}) \subseteq MOD_{PO\Sigma,PAX}$. Thus the parameter passing morphism f is correct, if the formal axioms hold in the actual specification.

Remark 22. A simple sufficient criterion for correctness of parameter passing is $PAX = \emptyset$. This was almost the case in our examples for parameterized object specifications, with the exception of some simple constraints for booleans and natural numbers.

4 Conclusion and Future Work

We have presented a proposal for parameterized object specifications allowing especially objects sorts in the parameter. As far as we know this has not been studied before. Our proposal for parameter passing works well on the syntactical level by

means of a pushout construction. It thereby establishes a link between well-known techniques from algebraic specification [EKT+84] and recent developments in information systems design with object-oriented concepts [GHC+94].
For the semantic level we have considered free constructions and corresponding forgetful functors. This last construction is rather tricky and calls for further study. It is open what kind of further properties this construction has. Another interesting question is under what conditions certain properties of the given actual object specification are preserved by the application of a parameterized object specification. For instance, one could ask whether all possible communications in the actual parameter are also possible in the resulting specification.

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

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

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

[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.

[EKT+84] H. Ehrig, H.-J. Kreowski, J. Thatcher, E. Wagner, and J. Wright. Parameter Passing in Algebraic Specification Languages. *Theoretical Computer Science*, 28:45–81, 1984.

[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.

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

[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, *Proc. 9th Workshop on Abstract Data Types (ADT'92)*, pages 1–29. Springer, Berlin, LNCS 785, 1994.

[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, *Proc. 10th Workshop on Abstract Data Types (ADT'94)*, pages 288–304. Springer, Berlin, LNCS 906, 1995.

[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.

[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.

[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.

[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.

[JHSS91] R. Jungclaus, T. Hartmann, G. Saake, and C. Sernadas. Introduction to TROLL – A Language for Object-Oriented Specification of Information Systems. In G. Saake and A. Sernadas, editors, *Information Systems – Correctness and Reusability*, pages 97–128. TU Braunschweig, Informatik Bericht 91-03, 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.

[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.

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