Proc. 4th Int. Conference on Conceptual Graphs (ICCS'96)
G. Ellis (Ed.), University of New South Wales, Sydney (1996)

Towards Object Visualization by Conceptual Graphs

Martin Gogolla
Bremen University
Informatics Department
Database Systems Group
D-28334 Bremen, Germany
e-mail: gogolla@informatik.uni-bremen.de

Abstract

The specification language TROLL *light* allows to describe the part of the world to be modeled in the information system design process as a community of concurrently existing and communicating objects. It follows the object paradigm by coherently describing structure as well behavior of conceptual objects. A TROLL *light* specification is divided into a signature part and an axiom part where formulas over the signature characterizing the object are formulated.

Conceptual graphs are employed to visualize object schemata as well as object instances. Thereby, we provide a uniform platform for representing static and dynamic properties of objects and reasoning about them. By giving appropriate graph substitution rules, conceptual structures are capable to express system dynamics as well.

1 Introduction

Conceptual graphs [Sow84] are designed to be a universe modelling language and reasoning formalism for different kinds of application areas in computer science. They are able to model data, functional, and dynamic aspects of computing systems. Furthermore, conceptual graphs can be used to form the basis for integrating such diverse description techniques as Entity-Relationship diagrams, finite state machines, Petri nets, or dataflow diagrams.

Here, we continue the tradition of approaches studying the relationship between conceptual graphs and object-orientation [Sow93a, Sow93b, WG93, KFKD94, Ell95]. We introduce an information system specification language based on the object-oriented paradigm. Our approach also has close connections to semantic networks [Leh92] and database design, especially classical semantic data models [HK87, PM88] like the Entity-Relationship model [Che76, Gog94]. Our motivation for using conceptual graphs lies in the aim of gaining a better understanding of complex specifications by visualizing them. Such an approach is well-tried nowadays and successfully employed by object-oriented methods like OMT [RBP+91] or the Coad-Yourdon approach [CY90].

But conceptual graphs have been studied also intensively in connection with the database field. In [CC93] conceptual graphs are employed for information system design. The paper shows how to translate NIAM schemata to conceptual graphs. The approach of [BC93] addresses data model translation and integration by means of a three-level functional architecture. The development of a foundation for deductive object-oriented database systems on the basis of specialized conceptual graphs is proposed in [WG93]. [BK93] studies the implementation of a conceptual graph layer on top of a relational database system. Information systems schema integration by means of conceptual graphs in the topic of [CE93]. There, possibilities of doing the integration task automatically are pointed out. A proposal to overcome the separation of queries and data in the relational data model is presented in [BCHL93]. An interesting idea to study dynamic properties specified with modal logic by means of conceptual graph structures is explained in [vdB93]. In [KFKD94] an extension of Prolog with respect to object-orientation called Prolog++ is defined. The approach of [BCHL93] is extended in [CH94] where a proposal to treat incomplete information and queries about queries is made. The ideas of [Ell95] are designed to improve the ideas of [Sow93a] with respect to the simplification of certain proofs. [Ell95] employs the object specification language Object-Z [DKRS91]. In contrast to the above mentioned papers we start from a formally defined textual language allowing general object descriptions and employ conceptual graphs for the visualization of both object descriptions and object states as well as object state transitions.

2 The Language TROLL *light*

TROLL *light* 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. As in TROLL object descriptions are called templates in TROLL *light*. Because of their pure descriptive nature templates may roughly be compared with the notion of class found in object-oriented programming languages. In the context of databases however, classes are also associated with class extensions so that we used a different notion. Templates show the following structure.

```
TEMPLATE name of the template
  DATA TYPES   data types used in current template
  TEMPLATES    other templates used in current template
  SUBOBJECTS   slots for sub-objects
  ATTRIBUTES   slots for attributes
  EVENTS       event generators
  CONSTRAINTS  restricting conditions on object states
```

```
    VALUATION    effect of event occurrences on attributes
    DERIVATION   rules for derived attributes
    INTERACTION  synchronization of events in different objects
    BEHAVIOR     description of object behavior by a CSP-like process
  END TEMPLATE
```

Roughly speaking, the `DATA TYPES` and `TEMPLATES` sections are the interfaces to other templates, the `SUBOBJECTS`, `ATTRIBUTES`, and `EVENTS` sections constitute the template signature, and in the remaining sections axioms concerning static (`CONSTRAINTS` and `DERIVATION`) and dynamic (`VALUATION`, `INTERACTION`, and `BEHAVIOR`) properties are specified.

To give an example for templates let us assume that we have to describe authors. For every author the name, the date of birth, and the number of books sold by year have to be stored. For the dynamic part we require that an author may change her name only once in her life. An appropriate TROLL *light* specification would hence be:

```
TEMPLATE Author
  DATA TYPES   String, Date, Nat;
  ATTRIBUTES   Name:string;BirthDate:date;SoldBooks(Year:nat):nat;
  EVENTS       BIRTH create(Name:string, BirthDate:date);
                     changeName(NewName:string);
                     storeSoldBooks(Year:nat, Number:nat);
               DEATH destroy;
  VALUATION    [create(N,D)] Name=N, BirthDate=D;
               [changeName(N)] Name=N;
               [storeSoldBooks(Y,NR)] SoldBooks(Y)=NR;
  BEHAVIOR     PROCESS authorLife1 =
                 ( storeSoldBooks -> authorLife1 |
                   changeName -> authorLife2 |
                   destroy -> POSTMORTEM );
               PROCESS authorLife2 =
                 ( storeSoldBooks -> authorLife2 |
                   destroy -> POSTMORTEM );
               ( create -> authorLife1 );
END TEMPLATE;
```

2.1 Data types section.

Data types are assumed to be specified with a data type specification language like SPECTRUM, ACT ONE, or OBJ3. With the `DATA TYPES` section the signature of data types is made known to the current template. For example, referring to `Nat` means that the data sort `nat`, operations like `+ : nat x nat -> nat`, and predicates like `<= : nat x nat` are visible in the current template definition. Note that we employ a certain convention concerning the naming of data types, templates and associated sorts. We use names starting with an upper case letter to denote data types and templates whereas the corresponding sort names start with a lower case letter.

2.2 Attributes section.

Attributes denote observable properties of objects. They are specified in the `ATTRIBUTE` section of a template by $a\,[\,\mathtt{(}s_1,\ldots,s_n\mathtt{)}\,]\,\mathtt{:}\,d$, where a is an attribute name generator, d is a sort expression determining the range of an attribute, and $s_1,\ldots,s_n$ $(n \geq 0)$ denote optional parameter sorts (data or object sorts). The sort expression d may be built over both data sorts and object sorts by applying predefined type constructors. We have decided to include the type constructors `tuple`, `set`, `bag`, `list`, and `union`. Of course other choices can be made. Thereby, complex attributes can be specified, e.g., data-valued, object-valued, multi-valued, composite, alternative attributes, and so on. The interpretation of all sort expressions contains the undefined element $\perp$, and therefore all attributes are optional by default. Attribute names may be provided with parameters. For example, by the declaration `SoldBooks(Year:nat):nat` a possibly infinite number of attribute names like `SoldBooks(1994)` is introduced. To underline the meaning of attribute parameters, optional parameter names like `Year` in `SoldBooks` may be added. We demand that in a given state of an object only a finite number of attributes takes values different from $\perp$ such that only these attributes have to be stored. Therefore, a parametrized attribute $a(s_1,\ldots,s_n) : s$ can also be viewed as an attribute $a : \mathtt{set}(\mathtt{tuple}(s_1,\ldots,s_n,s))$, but clearly the formerly inherent functional dependency would have to be described by an explicit constraint. The same works for parametrized sub-object constructors to be discussed later.

2.3 Events section.

Incidents possibly appearing in an object's life are modeled by events. Event names are given in the `EVENT` section by $e[\mathtt{(}s_1,\ldots,s_n\mathtt{)}]$, where e denotes an event generator and $s_1,\ldots,s_n$ $(n \geq 0)$ represent optional parameter sort expressions. Event parameters are used to define the effect of an event occurrence on the current state of an object (see the explanation of the `VALUATION` section below), or they are used to describe the transmission of data during communication (see the explanation of the `INTERACTION` section below). Special events in an object's life cycle are the `BIRTH` and `DEATH` events with which an object's life is started or ended. Several birth or death events may be specified for the same object. A template may have no death event, but we require it to have at least one birth event.

2.4 Valuation section.

The effect of events on attribute values is specified in the `VALUATION` section of a template definition by valuation rules, having the following form: [`{`*precond*`}`] `[`*event_term*`]` *attr_term* `=` *term*. Such a rule says that immediately after an event occurrence belonging to the event term *event_term*, the attribute given by the attribute term *attr_term* has the value deter-

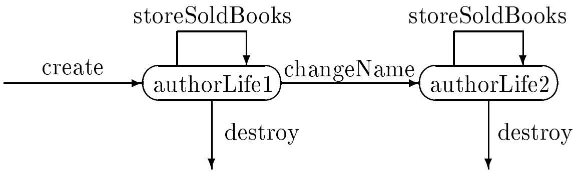

Figure 1: Behavior of authors

mined by destination *term*. The applicability of such a rule may be restricted by a formula *precond*, i.e., the valuation rule is applied only if the precondition is true. The precondition as well as both terms are evaluated in the state before the event occurrence. Thereby all these items may have free variables which however must appear in the event term. An event term consists of an event generator and a (possibly empty) list of variables representing the parameters of an event. By a concrete occurrence of such an event, these variables are instantiated with the actual event parameters in such a way that the precondition and the other terms can be evaluated.

To summarize we can say that a valuation is a proposition stating that after the occurrence of a certain event some attributes shall have some special values. The following frame rule is assumed: Attributes which are not caught by a suitable valuation rule of a given event remain unchanged. Before the birth event, all attributes of an object are assumed to be undefined. Thus if the event in question is a birth event, some attributes may remain undefined. An attribute can only be affected by local events, i.e., events which are specified in the same template in which the attribute is specified.

2.5 Behavior section.

In the `BEHAVIOR` section the possible event sequences in an object's life are restricted by means of behavior patterns. Roughly speaking, behavior patterns determine a finite non-deterministic machine called o-machine (o for object) in the sequel. However, regarding an object together with its attributes we shall in general get an infinite number of object states and not a finite one. By behavior patterns event sequences, which an object must go through, can be specified as well as event dependent branchings. We have used the keyword `POSTMORTEM` within the behavior patterns to denote that the object vanishes. In Fig. 1 the behavior of authors is visualized by the corresponding o-machine representation. Within an object description the behavior section may be missing. In that case event sequences are unrestricted, i.e., it is only required for event sequences to start with a birth event and possibly end with a death event. In addition to the mentioned possibilities, behavior patterns may include preconditions for event occurrences referring to the objects' attributes and sub-objects in order to restrict the possible event sequences further. Thus, edges in the o-machine may not only be labeled by event names but should also mention the appropriate precondition; in our example these preconditions are all `true`.

After dealing with the TROLL *light* features for simple objects we now turn to composite objects. In order to combine several authors in a higher-level object, classes are usually introduced as containers in object-oriented databases. In TROLL *light* an explicit class concept is not needed. Classes are viewed as composite objects instead, and therefore classes are described by templates as already mentioned before. However, means of describing the relationship between a container object and the contained objects must be added. This is done by introducing sub-object relationships denoting (exclusive) part-of relationships. The following example gives the format of container objects for authors.

```
TEMPLATE AuthorClass
  DATA TYPES    String, Date, Nat;
  TEMPLATES     Author;
  SUBOBJECTS    Authors(No:nat):author;
  ATTRIBUTES    DERIVED NumberOfAuthors:nat;
  EVENTS        BIRTH createClass;
                      addObject(No:nat,Name:string,BirthDate:date);
                      removeObject(No:nat);
                DEATH destroyClass;
  CONSTRAINTS   NumberOfAuthors<10000;
  DERIVATION    NumberOfAuthors=CNT(Authors);
  INTERACTION   addObject(N,S,D) >> Authors(N).create(S,D);
                removeObject(N) >> Authors(N).destroy;
END TEMPLATE;
```

2.6 Templates section.

Within the `TEMPLATES` section, other (existing) templates can be made known to the current template. We assume templates to induce corresponding object sorts. Hence referring to `Author` means that the object sort `author`, and the attributes and the event generators of `Author` are visible in `AuthorClass`.

2.7 Subobjects section.

An object of sort `authorClass` will hold finitely many author objects as private components or `SUBOBJECTS`. In order to be able to distinguish several authors from a logical point of view, an explicit identification mechanism is needed. One way to install such a mechanism would be to assign a unique name for each sub-object, e.g., `MyAuthor`, `YourAuthor`, ... Indeed, such a name allocation could be expressed in TROLL *light* as follows

```
SUBOBJECTS  MyAuthor:author; YourAuthor:author; ...;
```

But clearly, in the case of a large number of authors such a procedure is not practicable. A solution is given by parametrized sub-object constructors as shown in the example. As with parametrized attributes, a possibly infinite number of logical sub-object names like

`Authors(42)` are defined by the sub-object name declaration for `Authors`, but not the author objects themselves. The declaration only states that in context of an object of sort `authorClass`, author objects are identified by a natural number. In semantic data models the parameter `No` would be called a key. But the parameters need not be related to any attributes of the objects they identify. Each logical sub-object name corresponds to an object of the appropriate object sort. Analogously to attributes, we demand that in each state only a finite number of defined sub-objects exists.

2.8 Interaction section.

Interaction rules are specified in the `INTERACTION` part of the template definition. During the lifetime of an author class there will be events concerning the insertion or deletion of authors. In `AuthorClass`, the insertion of an author should always be combined with the creation of a new author object. In the template definition this is expressed by an event calling rule `addObject(N,S,D) >> Authors(N).create(S,D)`, where `N` denotes a natural number, `S` a string, and `D` a date. The general event calling scheme is expressed as [`{`*precond*`}`] [*src_obj* `.`]*src_event* `>>` [*dest_obj* `.`]*dest_event*. Such a rule states that whenever an event belonging to the source event *src_event* occurs in the source object denoted by *src_obj*, and the formula *precond* holds, then the event denoted by the destination event *dest_event* must occur simultaneously in the destination object denoted by *dest_obj*. If one of the object terms is missing, then the corresponding event term refers to the current object. The source object term is not allowed to have free variables, but the destination object term, the destination event term, and the precondition are allowed to have free variables, which however have to occur in the source event term. As already mentioned in the explanation of valuation rules, these free variables are instantiated by an event occurrence corresponding to the event term in such a way that the precondition and the other terms can be evaluated. In almost all cases the objects to be called for must be alive. The one and only exception is when a parent object calls for the birth of a sub-object. In this case it is even a requisite that the destination object does not exist already.

2.9 Constraints and derivation sections.

In addition to the mentioned language features TROLL *light* offers some further, more sophisticated concepts which will be mentioned only briefly: Possible object states can be restricted by explicit constraints, which are specified in the `CONSTRAINTS` section of a template. Constraints are given by closed formulas in the TROLL *light* query calculus, an offspring of an extended Entity-Relationship calculus [GH91]. For example, an author container of sort `authorClass` may only be populated by less than 10000 authors (as required above). Derived attributes can be specified by stating `DERIVATION` rules which, in fact, are closed terms of the TROLL *light* query calculus. In the example we declared the attribute `NumberOfAuthors` of `authorClass` as derived. We employed the function `CNT` which counts the elements of a multi-valued term. In the derivation rule `Authors` is regarded as a term

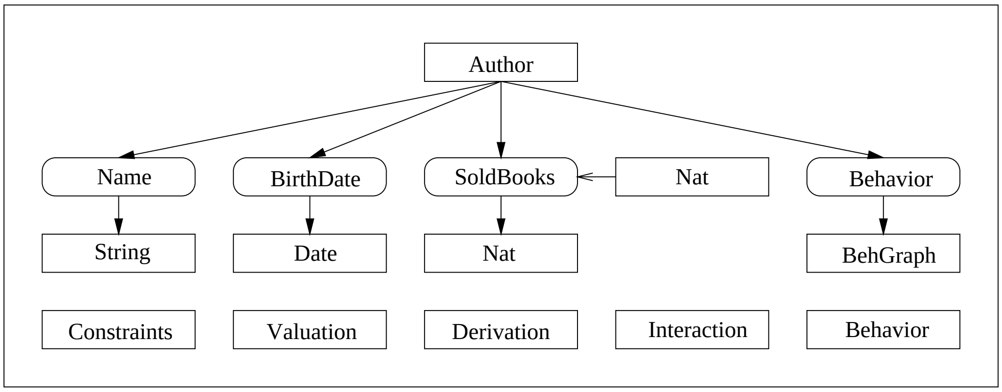

Figure 2: Conceptual graph for (part of) the author template

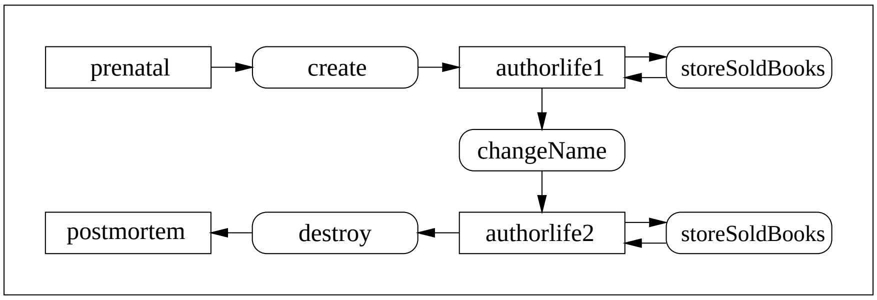

Figure 3: Conceptual graph for behavior states

of sort `set(tuple(nat,author))`. So `NumberOfAuthors` does not need to be set by explicit valuation rules.

Up to now we have only dealt with the description of objects (i.e., templates). In order to obtain object instances we must choose one template as the schema template, and one fixed object of the corresponding object sort as the schema object. All other objects in an object community will be sub-objects of this distinguished schema object. More details concerning TROLL *light* can be found in [GCH93, HCG94, CGH95]. In particular, [GCH93] discusses a graph-based semantics of the language as a state transition system in which a state transition is accompanied by a finite set of event occurrences which has to be closed against synchronization, i.e., interaction rules.

3 TROLL *light* and Conceptual Graphs

We now explain how TROLL *light* object description and their instances can be represented by conceptual graphs. We loosely follow the line of displaying objects by conceptual structures as it has been proposed in [Sow93a, Ell95].

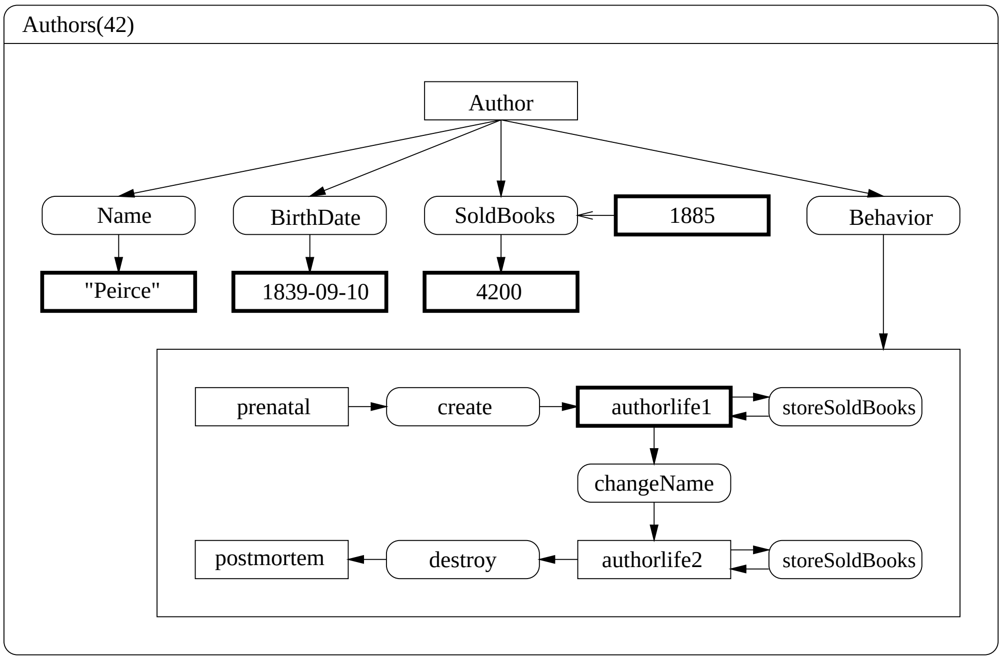

Figure 4: Conceptual graph for author instance

Templates: The "Author" template is visualized by a conceptual graph in Fig. 2. The concepts depicted by rectangles correspond to the sorts induced by the template. The conceptual relations represented by ovals give the explicit object attributes and the behavior state which can be regarded as an implicit attribute. The behavior graph for the behavior state is a complex conceptual structure (in contrast to the more simpler graphs for strings or natural numbers).

Behavior patterns: Figure 3 describes the author behavior in more detail. Concepts correspond to behavior states and conceptual relations to events. Here, we have an example of a very simple behavior graph. In general, we will have not only event names as conceptual relations but in addition there will be preconditions (formulas of the calculus mentioned above) for event occurrences. Thus the general conceptual graphs for behavior patterns will be more involved.

Template instance: In Fig. 4 an author object instance identified from outside by "Authors(42)" (shown in the upper line) is displayed as a conceptual graph. The fat parts of the graph emphasize the state dependent part, i.e., the value of the respective attributes: the object has the "Name" value "Peirce"; the date of birth is "1839-09-10"; the object has sold "4200" books in year "1885", and its behavior state shows that (at least) the event "create" has occurred.

Valuation rules: The valuation rule for the event "storeSoldBooks" is given in Fig. 5. The

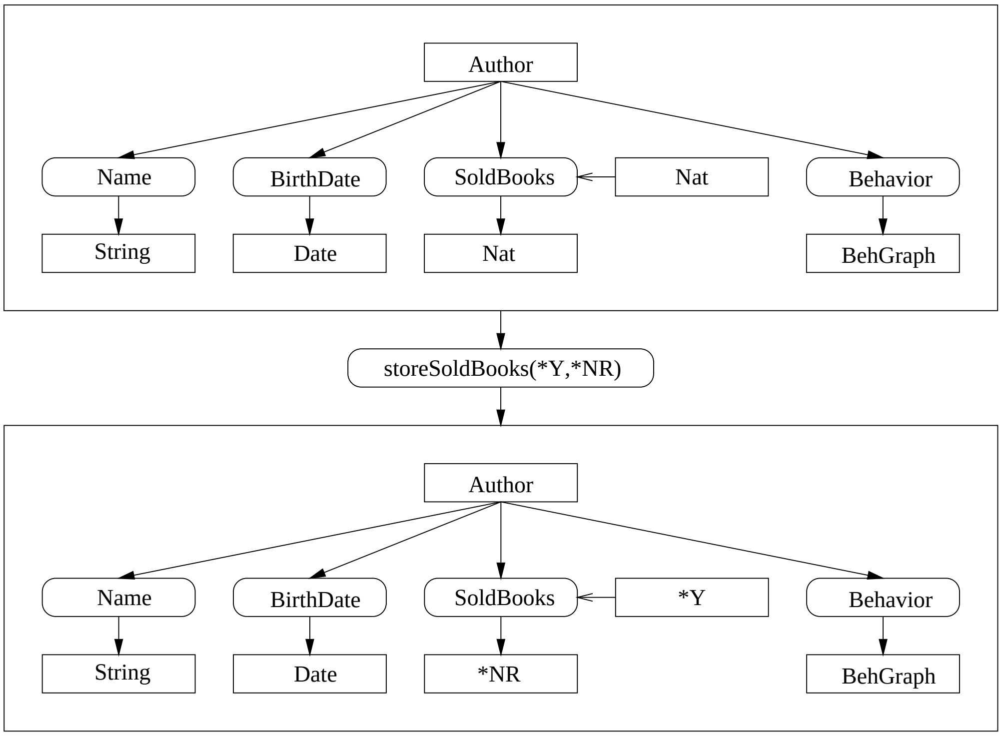

Figure 5: Conceptual graph for valuation rule of “storeSoldBooks”

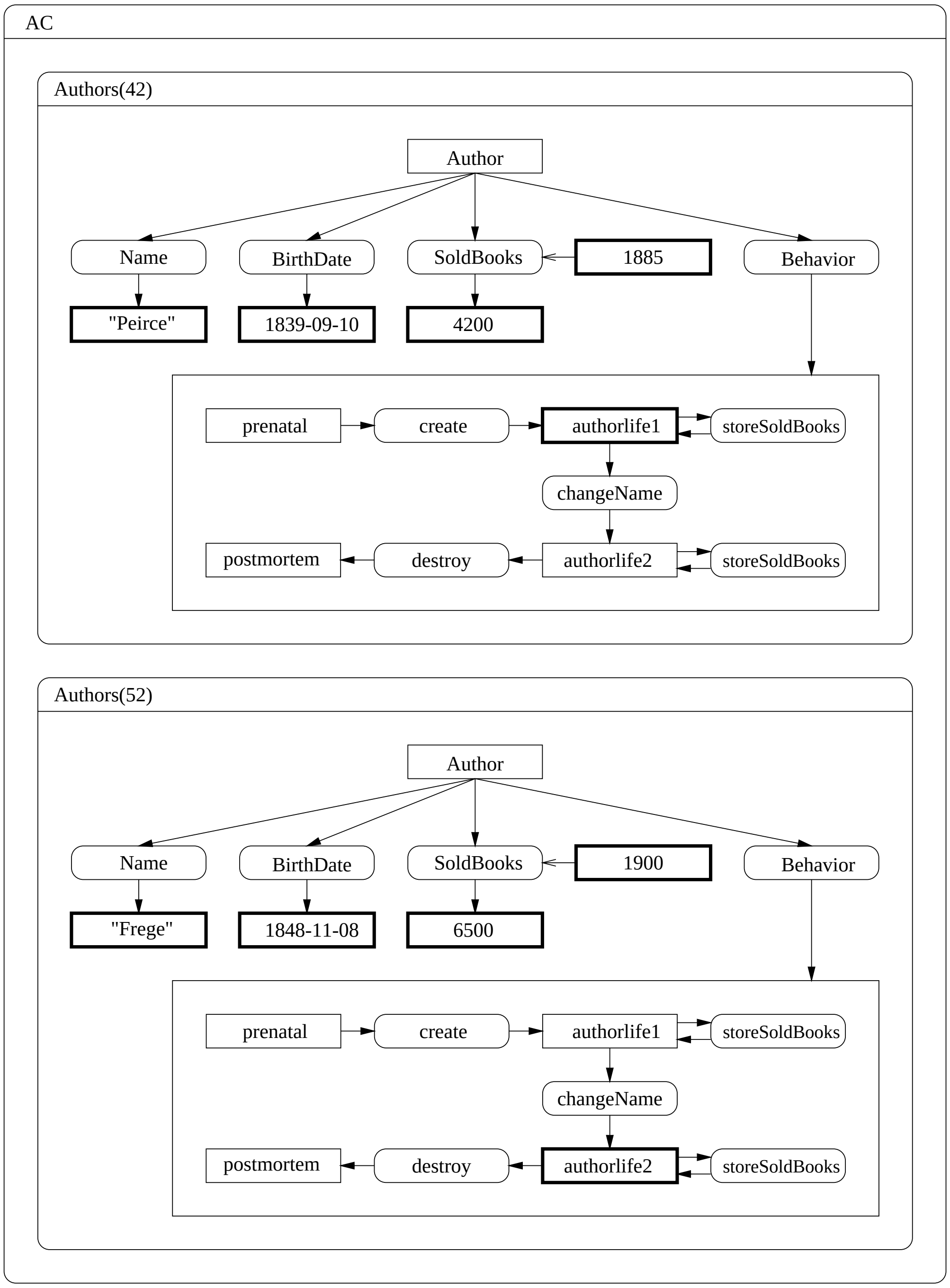

Figure 6: Conceptual graph for authorclass instance

rule is given as a conceptual relation between two complex object graphs. It adds a new "SoldBooks" node and two "Nat" nodes corresponding to the year and the number of sold books in that year; their values are determined by the parameters of the occurring event.

Objects with sub-objects: Figure 6 shows a complex authorclass object with sub-objects identified by "AC". It embodies two author objects identified by "Authors(42)" and "Authors(52)". The inner two objects are displayed with their distinguishing attribute values. In this situation, conceptual graphs are perfectly suited to intuitively express the formal semantics of the TROLL *light* "subobjects" clause.

We cannot explain how all the other parts of TROLL *light* specifications are represented by conceptual graphs. But all parts, especially integrity constraints and derivation rules, which are close to first-order calculus formulas, can be depicted in the spirit explained in [Sow84, Sow93b].

4 Conclusion and Future Work

Here, we have presented a sketch of our object description language TROLL *light* and have explained how the specifications and the corresponding instances can be visualized by conceptual graphs. However, up to now we have not considered implementation aspects, and we have not studied how our development environment benefits from such an approach. This is subject to future research.

Acknowledgements

Our research activities have been constructively criticized by our former KORSO colleagues, especially Rudolf Herzig.

References

[BC93] G. Berg-Cross. Using Conceptual Structures to Translate Data Models: Concepts, Context and Cognitive Processes. In H.D. Pfeiffer and T.E. Nagle, editors, *Proc. 7th Annual Workshop on Conceptual Structures*, pages 171–187. Springer, Berlin, LNCS and LNAI 754, 1993.

[BCHL93] C. Boksenbaum, B. Carbonneill, O. Haemmerle, and T. Libourel. Conceptual Graphs for Relational Databases. In G.W. Mineau, B. Moulin, and J.F. Sowa, editors, *Proc. 1st Int. Conf. on Conceptual Structures (ICCS'93)*, pages 142–161. Springer, Berlin, LNCS and LNAI 699, 1993.

[BK93] B. Bowen and P. Kocura. Implementing Conceptual Graphs in a RDBMS. In G.W. Mineau, B. Moulin, and J.F. Sowa, editors, *Proc. 1st Int. Conf. on Conceptual Structures (ICCS'93)*, pages 106–125. Springer, Berlin, LNCS and LNAI 699, 1993.

[CC93] L. Cambell and P. Creasy. A Conceptual Graphs Approach to Information Systems Design. In H.D. Pfeiffer and T.E. Nagle, editors, *Proc. 7th Annual Workshop on Conceptual Structures*, pages 46–58. Springer, Berlin, LNCS and LNAI 754, 1993.

[CE93] P. Creasy and G. Ellis. A Conceptual Graphs Approach to Conceptual Schema Integration. In G.W. Mineau, B. Moulin, and J.F. Sowa, editors, *Proc. 1st Int. Conf. on Conceptual Structures (ICCS'93)*, pages 126–141. Springer, Berlin, LNCS and LNAI 699, 1993.

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

[CH94] B. Carbonneill and O. Haemmerle. Standardizing and Interfacing Relational Databases Using Conceptual Graphs. In W.M. Tepfenhart, J.P. Dick, and J.F. Sowa, editors, *Proc. 2nd Int. Conf. on Conceptual Structures (ICCS'94)*, pages 311–330. Springer, Berlin, LNCS and LNAI 835, 1994.

[Che76] P.P. Chen. The Entity-Relationship Model – Towards a Unified View of Data. *ACM Trans. on Database Systems*, 1(1):9–36, 1976.

[CY90] P. Coad and E. Yourdon. *Object-Oriented Analysis*. Yourdon Press/Prentice Hall, 1990.

[DKRS91] R. Duke, P. King, G. Rose, and G. Smith. The Object-Z Specification Language. In *Proc. Technology of Object-Oriented Languages and Systems (TOOLS'91)*, pages 465–483. Prentice-Hall, 1991.

[Ell95] G. Ellis. Object-Oriented Conceptual Graphs. In G. Ellis, R. Levinson, W. Rich, and J.F. Sowa, editors, *Proc. 3rd Int. Conf. on Conceptual Structures (ICCS'95)*, pages 144–157. Springer, Berlin, LNCS and LNAI 954, 1995.

[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.

[GH91] M. Gogolla and U. Hohenstein. Towards a Semantic View of an Extended Entity-Relationship Model. *ACM Trans. on Database Systems*, 16(3):369–416, 1991.

[Gog94] M. Gogolla. *An Extended Entity Relationship Model. Fundamentals and Pragmatics.* Springer, Berlin, LNCS 767, 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.

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

[KFKD94] A. Kabbaj, C. Frasson, M. Kaltenbach, and J.-Y. Djamen. A Conceptual and Contextual Object-Oriented Logic Programming: The PROLOG++ Language. In W.M. Tepfenhart, J.P. Dick, and J.F. Sowa, editors, *Proc. 2nd Int. Conf. on Conceptual Structures (ICCS'94)*, pages 251–274. Springer, Berlin, LNCS and LNAI 835, 1994.

[Leh92] F. Lehmann. *Semantic Networks in Artificial Intelligence.* Pergamon Press, Oxford, 1992.

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

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

[Sow84] J.F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, Reading (MA), 1984.

[Sow93a] J.F. Sowa. Logical Foundations for Representing Object-Oriented Systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 5, 1993.

[Sow93b] J.F. Sowa. Relating Diagrams to Logic. In G.W. Mineau, B. Moulin, and J.F. Sowa, editors, *Proc. 1st Int. Conf. on Conceptual Structures (ICCS'93)*, pages 1–35. Springer, Berlin, LNCS and LNAI 699, 1993.

[vdB93] H. van den Berg. Modal Logics for Conceptual Graphs. In G.W. Mineau, B. Moulin, and J.F. Sowa, editors, *Proc. 1st Int. Conf. on Conceptual Structures (ICCS'93)*, pages 411–429. Springer, Berlin, LNCS and LNAI 699, 1993.

[WG93] V. Wuwongse and B.C. Ghosh. Towards Deductive Object-Oriented Databases Based on Conceptual Graphs. In H.D. Pfeiffer and T.E. Nagle, editors, *Proc. 7th Annual Workshop on Conceptual Structures*, pages 188–205. Springer, Berlin, LNCS and LNAI 754, 1993.