Nuclear Instruments and Methods in Physics Research A 352 (1994) 375–378
North-Holland

NUCLEAR
INSTRUMENTS
& METHODS
IN PHYSICS
RESEARCH
Section A

Fundamentals of object-oriented information systems specification and design: the OBLOG/TROLL approach †

Hans-Dieter Ehrich *

Abteilung Datenbanken, Technische Universität, Postfach 3329, D-38023 Braunschweig, Germany

A survey of concepts for an information system specification is given, based on the viewpoint that an information system is a community of interacting objects. Objects are self-contained units of structure and behavior capable of operating independently and cooperating concurrently. The approach integrates concepts from semantic data modeling and concurrent processes, adopting structuring principles partly developed in the framework of object-orientation and partly in that of abstract data types. The languages OBLOG and TROLL are based on these concepts and their use is illustrated by examples.

1. Introduction

Conventional information system design starts with separating data from operations, designing each with its own collection of concepts, methods, tools – and people; conceptual modeling for the information structure and program design for the application programs. This separation carries through until the final implementation: data are collected in databases and managed with database management systems, and application programs accessing the database are implemented in programming languages. This approach tends to suffer from a problem known as *impedance mismatch*, since the basic paradigms underlying databases and programs – modeling, design, languages and systems – do not fit; there are incompatible type systems, data formats, operation modes, etc. Recent 4th-generation systems encapsulate and hide the problem to some extent, but do not really remedy the situation.

The object-oriented paradigm promises to overcome the problem: a system is viewed as a community of interacting objects, each incorporating data and operations. While we still have object-oriented programming languages incompatible with object-oriented database systems, ideas and approaches seem to converge towards homogeneous software systems dealing with both data and operations in a unified way.

Viewing the system as a community of interacting objects does not solve all problems. Beyond the object concept, abstraction and structuring principles are needed, together with languages and methods to work with them. In addition, we need a sound theoretical underpinning that provides a basis for giving formal semantics and proof systems to languages so that relevant system properties can be formally stated and verified.

The ideas put forward in this paper have been mainly developed in the ESPRIT Working Group ISCORE, employing objects as a unifying concept and aiming at a conceptually seamless methodology from requirements to implementation [1–4].

The TROLL language development project at TU Braunschweig is based on OBLOG which originated in 1986 as an object-oriented specification language design effort for information systems [5,2]. Meanwhile, OBLOG is being developed into a commercial product by the OBLOG software company in Lisbon. Cooperating academic research is going on with INESC Lisbon [1,4,6]. At TU Braunschweig, TROLL is being developed along with an operational dialect called TROLL *light* [7].

These activities have been greatly influenced by related work on specification languages and theoretical foundations. For brevity, we mention only [3,8,9].

2. Basic concepts

An *information system* is a reactive system with a database and application programs. In object-oriented view, an information system is a community of interacting objects where an object is a unit with an immutable identity, encapsulating structure and behavior. In more technical terms, an object is a process endowed with data.

Usually, a system will handle many objects with

* Corresponding author.

† The work reported here was partly supported by the EC under ESPRIT BRA WG 3023/6071 ISCORE and WG 3264/6112 COMPASS, by DFG under Sa 465/1-1 and Sa 465/1-2, by BMFT in the KORSO project, and by the OBLOG Software company.

SSDI 0168-9002(94)00032-3

"the same" structure and behavior. A generic pattern of structure and behavior is called an object *template*. A template is like a type. An object *class* is something different, as a class has a dynamic population; a time-varying collection of member objects, all with the same template. The specification of an object community is called a *schema*. It consists of templates for objects and classes and several kinds of relationships between templates, such as roles, generalization, aggregation, interaction, and interfaces.

3. Templates

For specifying the following examples of templates, we use a TROLL-like pseudocode, omitting technical details.

3.1. Example

We give an example of a PERSON template, describing persons with names, capable of entering and leaving the scene, working, eating, getting hungry, etc. After a person has entered, she is given a name on entry, and she is not hungry. After work, she gets hungry. She has the choice of eating fish or meat. After eating fish, she is still hungry. After eating meat, she is full. She may eat only if she is hungry, and she may leave the scene only if she is not hungry. If she is hungry, she must eat meat some time.

```
template PERSON
  data types ...;
  attributes
    name:text;
    hungry:{yes,no};
  actions
    birth enter(n:text);
    eat(f:{fish,meat});
    work;
    death leave;
  valuation ...;
    [enter(n)]name=n;
    [enter(n)]hungry=no;
    [work]hungry=yes;
    [eat(fish)]hungry=yes;
    [eat(meat)]hungry=no;
    ...
  behavior
    permission ...;
    {hungry=yes}eat(f);
    {hungry=no}leave;
    ...
    obligation ...;
    {hungry=yes} ⇒ eat(meat);
    ...
end template PERSON.
```

Based on a conventional data signature, a *template signature* consists of attribute generators and action generators. Attributes are constant terms with an attribute generator at its root, and actions are constant terms with an action symbol at its root.

A template specification consists of a template signature and a set of axioms in template logic which is a linear temporal logic adapted to the needs at hand. We cannot go into detail here.

The interpretation structures of our template specification logic are life cycles, i.e. sequences of situations where a situation describes the current values of attributes, currently enabled and occurring actions, etc.

The meaning of a given template specification is the set of life cycles satisfying all given formulae. Adopting the life cycle model, a *process* is precisely this: a set of life cycles. Thus, the semantics of a template specification is a process, namely the process describing the most liberal behavior permitted by the axioms.

4. Template relationships

Templates can be related by the following relationships.

Role: a role is a temporary specialization. As an example, we might describe persons in their special roles as patient, employee, etc.

Generalization: A generalization is the reverse of a specialization. As an example, persons and companies may be generalized to customers.

Aggregation: Component objects may be integrated into complex objects. As an example, a team may be aggregated from a coach and a list of players.

Communication: Within an aggregated object, an action in one component may *call* actions in the same or other components. This means that the called actions happen simultaneously whenever the calling action happens.

Interface: An interface restricts the view to an object by hiding properties. As an example, the customer view of an automatic teller machine (ATM) is restricted from the bank view. Interfacing may introduce nondeterminism; the customer may observe different reactions from the ATM after performing exactly the same manipulation sequences, due to bank actions which are invisible to him.

Let us have a closer look at roles. Referring to example 3.1, we specify the patient concept as a role that a person can temporarily play. The life cycle segment between entering and leaving a role is called a *phase*. During a life cycle, an object may enter and leave several phase of the same role.

4.1. Example

A PATIENT has a temperature as a local attribute which is only visible when she is a patient. Similarly,

there is a local massage action. The local fall-ill and recover actions serve as entry into and exit from the role, respectively.

```
template PATIENT
 role of PERSON
 attributes
   temp:[35..42];
 actions
   entry fall-ill;
   massage;
   exit recover;
   ...
end template PATIENT.
```

We note in passing that a conventional specialization is a special case of a role, namely a permanent role entered with creation and left with destruction of an object.

The semantics of roles can also be described by means of formal logic but we cannot go into details here.

The PATIENT template in example 4.1 does not stand alone, it refers to the PERSON template in example 3.1. In PATIENT, all items of PERSON are visible as if PERSON would be a textual part of PATIENT.

In general, a template with roles defines a union template into which all role templates are textually embedded. The embedding preserves structure and behavior; it is an example of a *template specification morphism*. Specification morphisms make *substitution inheritance* precise: wherever a PERSON template is expected as a parameter, a PATIENT template can be submitted, to be adapted via the fitting morphism PERSON ↪ PATIENT.

On the semantic level, we have a corresponding projection reducing patient life cycles to person life cycles; from each patient situation in each patient life cycle, we omit the special patient items, retaining only the items visible for persons. This projection map from patient to person processes makes the idea of ISA *inheritance* precise; a patient "is a" person at the same time.

Also generalization, aggregation, and interfaces can be made precise with template specification morphisms and their semantic reduction maps, but we cannot go into details here. Another important subject we cannot discuss here is object communication. TROLL and OBLOG employ *action calling* which is a directed and simultaneous way of communication.

Generalization is the reverse of specialization, and so there is also a dynamic version. An example is the role of customer shared by persons, companies, communities, cities, states, etc. We need to elaborate on the role concept to allow for shared roles, expressing dynamic generalization as well as dynamic specialization.

5. Concluding remarks

Of course, this short note does not give a complete description of TROLL, so we have concentrated on very few aspects and simple examples. The interested reader is referred to the language reports [10,7].

The TROLL languages are based on concepts from semantic modeling, algebraic specification and specification of reactive systems, combining the advantages of these approaches. They offer a variety of structuring mechanism for specification, so that system specifications can be constructed from components that can be analysed locally. Language features, not discussed in this paper, give support for data type specification for attribute values, derived attributes, derived actions, derived components, integrity constraints, initiative, explicit process description, etc. New language features are under discussion including concepts such as implementation (sometimes called reification or refinement), parameterization, modules and libraries, which would support reuse by modularizing the system architecture.

The OBLOG language incorporates a workable subset of the TROLL concepts under a graphical interface. In a sense, the TROLL project serves as an academic playground where language concepts and implementations are being explored. Features which prove useful as well as implementable and supportable may eventually find their way into commercial utilization.

Acknowledgements

We gratefully acknowledge inspirations, discussions, suggestions and criticisms that we have received within the ISCORE project and the Braunschweig information systems group. Special thanks are due to Amilcar and Cristina Sernadas and their group at INESC, their influence on our work has been great. We also appreciate contributions by Gunter Saake and Martin Gogolla who are in charge of the TROLL and TROLL *light* projects, respectively. Close cooperation with Grit Denker and Ralf Jungclaus on recent papers on the subject is gratefully acknowledged.

References

[1] H.-D. Ehrich, G. Saake and A. Sernadas, in: Proc. 2nd Workshop of Informationssysteme und Künstliche Intelligenz: Modellierung, Ulm, Germany (Springer, Berlin, 1992) pp. 1–19.

[2] A. Sernadas and H.-D. Ehrich, in: R. Meersman, W.

Kent and S. Khosla, eds., Object-Oriented Databases: Analysis, Design and Construction, Proc. 4th IFIP WG 2.6 Working Conference DS-4, Windermere, UK, (North-Holland, Amsterdam, 1991) pp. 39–70.

[3] A. Sernadas and J. Fiadero, Data and Knowledge Engineering 6 (1991) 479.

[4] G. Saake, R. Jungclaus and H.-D. Ehrich, in: J. de Meer, V. Heymer and R. Roth, eds., Proc. Open Distributed Processing, Berlin, 1991, IFIP Transactions C: Communication Systems, vol. 1 (North-Holland, Amsterdam, 1992) pp. 99–121.

[5] A. Sernadas, C. Sernadas and H.-D. Ehrich, in: P.M. Stoecker and W. Kent, eds., Proc. 13th Int. Conf. on Very Large Databases VLDB'87 (VLDB Endowment Press, Saratoga (CA), 1987) pp. 107–116.

[6] H.-D. Ehrich, G. Denker and A. Sernadas, in: M.-C. Gaudel and J.-P. Jouannaud, eds., Proc. TAPSOFT'93: Theory and Practice of Software Development, LNCS 668 (Springer, Berlin, 1993) pp. 453–467.

[7] S. Conrad, M. Gogolla and R. Herzig, Informatik-Bericht 92-02, TU Braunschweig, 1992.

[8] J.A. Goguen and J. Meseguer, in: B. Shriver and P. Wegner, eds., Research Directions in Object-Oriented Programming (MIT Press, 1987) pp. 417–477.

[9] L. Rapanotti and A. Socorro, Technical Report PRG-TR-28-92, Programming Research Group, Oxford University Computing Lab, 1992.

[10] R. Jungclaus, G. Saake, T. Hartmann and C. Sernadas, Informatik-Bericht 91-04, TU Braunschweig, 1991.