Hans-Dieter Ehrich, Joseph A. Goguen,
Amilcar Sernadas (editors):

Foundations of Information Systems Specification and Design

Dagstuhl-Seminar-Report; 35
16.-19.3.9 (9212)

ISSN 0940-1121

Tel.: +49-6871 - 2458
Fax: +49-6871 - 5942

Das Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI) ist eine gemeinnützige GmbH. Sie veranstaltet regelmäßig wissenschaftliche Seminare, welche nach Antrag der Tagungsleiter und Begutachtung durch das wissenschaftliche Direktorium mit persönlich eingeladenen Gästen durchgeführt werden.

Bezugsadresse: Geschäftsstelle Schloß Dagstuhl
Informatik, Bau 36
Universität des Saarlandes
W - 6600 Saarbrücken
Germany
Tel.: +49 -681 - 302 - 4396
Fax: +49 -681 - 302 - 4397
e-mail: office@dag.uni-sb.de

Dagstuhl Seminar 9212

Foundations of Information Systems Specification and Design

16–19 March 1992

organized by

Hans-Dieter Ehrich
Abteilung Datenbanken, Technische Universität, Postfach 3329
W–3300 Braunschweig, GERMANY

Amilcar Sernadas
Computer Science Group, INESC, Apartado 10105
1017 Lisbon Codex, PORTUGAL

Joseph A. Goguen
Computing Laboratory, Programming Research Group
11 Keble Road, Oxford OX1 3QD, GREAT BRITAIN

About the Workshop

Information systems are reactive systems with a (typically large) data or knowledge base. Thus, information systems specification and design brings areas together which have developed separately so far: reactive systems design emphasizing processes and concurrency, database design emphasizing conceptual and "logical" data modeling, and knowledge base design emphasizing knowledge representation and reasoning.

Whereas single aspects in these areas are well understood, there is no coherent conceptual basis for the entire spectrum of information systems design. A better understanding of fundamental issues, however, is essential for progress in this field.

The purpose of the workshop was to bring together experts who have made substantial contributions to foundations in one or more of these areas, covering semantics, logics and proof theory, language features and methodological issues. Unifying approaches addressing aspects of several areas were especially looked for. Among these, the object–oriented paradigm and its theoretical foundation was a major focus of the workshop, but alternative approaches were also well represented.

Participants

Egidio Astesiano, University of Genova
Catriel Beeri, The Hebrew University of Jerusalem
Joachim Biskup, Universität Hildesheim
Rod M. Burstall, University of Edinburgh
Jun-Hee Cho, ETRI Daejon, Republic of Korea
Felix Costa, INESC Lisbon
Rolf A. de By, University of Twente
Robert Demolombe, CERT/DERI Toulouse
Hans-Dieter Ehrich, Technische Universität Braunschweig
Gregor Engels, Leiden University
Jose L. Fiadeiro, IST Lisbon
Joseph A. Goguen, University of Oxford
Ralf Jungclaus, Technische Universität Braunschweig
Georg Lausen, Universität Mannheim
Udo W. Lipeck, Universität Hannover
Tom S. E. Maibaum, Imperial College London
Grant Malcolm, University of Oxford
Rainer Manthey, ECRC München
Gunter Saake, Technische Universität Braunschweig
Isidro Ramos Salavert, Universidad Politecnica de Valencia
Amilcar Sernadas, INESC Lisbon
Arne Sølvberg, University of Trondheim
T. H. Tse, The University of Hong Kong
Reind Van de Riet, Vrije Universiteit Amsterdam
Roel Wieringa, Vrije Universiteit Amsterdam
Jeannette Wing, MIT Cambridge MA
David A. Wolfram, University of Oxford

Contents

D-oids and semantics of method expressions

Egidio Astesiano
DISI-Dipartimento di Informatica e Scienze dell'Informazione
University of Genova, Viale Benedetto XV 3, I-16132 Genova, ITALY

astes@igecuniv.bitnet

(Joint work with Elena Zucca)

Abstract

We introduce a formal model, dynamic structures (also called d-oids), which can play for systems of dynamic objects the role played by algebras for modelling static data types. D-oids may model classes of objects as universes of dynamic systems. An instant configuration of a system is formalized as an instant algebra, which describes, together with the auxiliary static values, the current states of the objects existing at that time. Passing from an instant algebra to another is the effect of a method. D-oids form a category and it is shown how some intuitive equivalences between classes can be modelled by isomorphisms. We have recently added two features to the approach, namely the definition of a reduct operation, which can model instantiated inheritance, and the construction of a free d-oid over a set of generators. It is shown how the free d-oid is the kernel of a language of method expressions, whose semantics is given easily in a compositional way. Also recursive method definitions are allowed and thus also the use of self is semantically formalized. In the above semantics a central role is played by a notion which is most distinguishing our approach: a method applied to an instant algebra and a tuple of arguments gives not only a target instant algebra (the new state) and possibly a value, but also a tracking map which shows how every element is transformed, thus also keeping track of the identities of the objects in a rather abstract way.

Algebraic Specifications for OODB's – Are They Useful?

Catriel Beeri
Department of Computer Science, The Hebrew University of Jerusalem
Givat RAM, 91904 Jerusalem, ISRAEL

beeri@cs.huji.ac.il

Abstract

The last decade has seen a succession of database models and languages. Naturally, one wonders whether a general framework for treating these exists. The talk presents the idea that a database has several (actually two) components. The domains and operations constitute an underlying data algebra, or type system. Collections of elements from it are the database contents. Both the data algebra and the database contents can be specified by algebraic specifications, using initial model semantics. In particular, the initial model semantics includes the CWA and the minimal model in the Herbrand universe semantics used in relational and deductive databases as special cases.

Now, it turns out that one can have predicate-based and algebraic query language paradigms that can work with any type system. The semantics of expressions/programs in such langauges can be specified using the same approach. The talk presents some results about the relationships between these paradigms. The talk concludes with observing that

- although initial model semantics was the starting point, it seems that a full tretament of these language paradigms necessitate going beyond initial model semantics;
- although the ideas of algebraic specification guided the research, specifications are not really used, except to provide existence and uniqueness theorems for the semantics of the languages;
- it does not seem that the approach will be useful for update languages.

On The Design Theory for Database Schemes

Joachim Biskup
Institut für Informatik, Universität Hildesheim
Postfach 130, W-3200 Hildesheim, GERMANY

biskup@informatik.uni-hildesheim.de

Abstract

Design theory should help to understand how the (static) structural aspects, as declared in a database schema, determine the (dynamic) operational aspects (mostly not explicitly dealt with in traditional data models). This rough idea is made more precise for a specific example: relational database schemes with functional, inclusion, and exclusion dependencies. These dependencies allow to express keys and unique relationships, hierarchies and references (foreign keys), and partitions, respectively, and thus their expressiveness should be available for any advanced data model. We formally define the notion of an attribute set X being an object by requiring (informally) that 1) X-values should be unique, 2) new X-values can be inserted without violating dependencies 3) existing X-values can be deleted without affecting other objects. Thus the notion of an object deals with an operational aspect. We can relate this operational aspect with a series of properties of dependencies that are purely structural. For instance we have a theorem stating that attribute set $X \subseteq R$, R a relation scheme, is an object iff (informally) R is not referencing, R is not involved in partitioning, R is a Boyce/Codd Normal Form, and X is a unique minimal key of R. Finally we define a notion of Object Normal Form, characterize it in terms of dependencies, show its undecidabily in general, and point out an interesting though decidable special case. As a perspective, future work on using results of the presented kind for concretely coding constructors and deconstructor for object class specifications is mentioned.

Proofs in Constructive Logic

Rod M. Burstall
Dept. of Computer Science, University of Edinburgh
Mayfield Road, Edinburgh EH9 3JZ, UK

rb@dcs.ed.ac.uk

Abstract

Lego is a proof assistent written by Randy Pollack, implementing an extension of the Calculus of Constructions with Σ-types, i.e., dependent part types. The language is very small (few constructs) and can represent specifications, (functional) programs and proofs directly. It provides Higher Order Intuitionistic Logic with Natural Deduction proofs which are developed interactively top down. The only search is in the unification algorithm which is able to expand definitions to achieve unification.

Luo has used the Σ-types to develop a machine checked theory in Lego for concepts including specifications, models, refinements, parameterized specifications and operations on specifications.

Claire Jones has developed the definition of reals from rationals and also completion of a metric space.

The Lego system is available from Edinburgh freely by FTP.

Algebraic Theory of Transition Systems Implementation

Felix Costa
INESC, Apartado 10 105, Rua Alvez Redol 9, 7a
P-1017 Lisboa Codex, PORTUGAL

fgc@inesc.pt

Abstract

A semantic domain based on labelled transition systems is proposed for object-oriented concepts supporting implemen tations of objects over objects.The implementation of the abstract object is introduced via a derivation/implementa tion of the attributes and the events of the abstract object on top of the attributes and events of the ground object.

The theory of implementation of labelled transition systems is such that: (a) implementation morphisms compose, and (b) the implementation of a composite transition system is the composition of the implementations of its parts.

A Functional Database Specification Language with Subtyping

Rolf A. de By
Dept. of Computer Science, University of Twente
Postbus 217, NL-7500 AE Enschede, THE NETHERLANDS

deby@cs.utwente.nl

Abstract

An overview is presented of the TM/FM-project currently running in our research group. TM is a conceptual database specification language that is based on a formal model FM. TM is a functional language with a strict typing and subtyping discipline, which is given a formal semantics by the typed lambda calculus nature of its underlying model FM.

In the first part of the presentation, notions like types, expressions, subtypes, classes and methods are discussed. A rather general approach is shortly discussed as how to obtain a formal semantics of types and expressions that obeys the intuitive requirement that, in the semantics, subtyping leads to set inclusion, and the typing relation leads to set membership. A notion of 'minimal type' and its usefulness is then discussed. These results are due to Balsters & Fokkinga, Theoretical Computer Science 87: 81–96.

In the middle part of the talk, focus shifts towards a specification methodology for object-oriented databases, and the main interest here is the definition of a *database universe*. It is shown that, by going from object (types) via table (types) to the database (type) a natural specification sequence is defined, that gives rise to the identification of objects at different levels of granularity. For each of these levels constraints and methods can be defined. The issue of inheritance of method bodies is raised.

In the last part of the presentation, these lessons are applied for transaction specification, in which the database is viewed as a single, albeit complex, object. The advantages and disadvantages of a functional approach are then discussed, and specific so-called 'generalizing primitives' are defined that enable the language user to bridge the gaps between the different levels of granularity for method specification. It turns out that issues like object generation can, in a functional approach, only be described at the database level, and this gives more insight in the complexity of such operations.

New Deduction Techniques to Retrieve Information in Data and Knowledge Bases

Robert Demolombe
CERT/DERI, Av. Edouard Belin 2, F-31055 Toulouse, FRANCE

demolomb@tls-cs.cert.fr

Abstract

At the very beginning deduction techniques have been developped in the Theorem Proving field to answer the following question: given a theory T and a formula Q, does Q logically follow from T?

Then these techniques were extended in the field of Logic Programming to answer the question: what is the set of substitutions s such that Q.s logically follows from T?

We have developped new deduction techniques to answer a new kind of question: what are the logical consequences of T satisfying a given property P? The answer to such kind of query is neither a truth value nor a set of substitutions, but a set of formulas.

We have presented deduction strategies, called GASP and GALP, to retrieve all the consequences, in clausal form, that contain an instance of a given literal Q. These consequences are of the form Q$\vee$X and can be understood as conditional answers when they are written in the form : Q$\leftarrow \neg$X. We have shown that GASP is always more efficicent than GALP, but it may not terminate when T contains recursive definitions. For this reason a combination of GASP and GALP, called GRASP, has been designed that combines positive features of both strategies. GRASP allows to retrieve only ground conditional answers, and it was proven that GRASP always terminates. All these strategies are formally defined using meta-predicates and meta-rules.

Finally, a specific inference rule called P-inference, has been defined to derive all, and only all, T consequences satisfying a given property P, and it has been proved that if P is a "backward" property, and if property P is preserved through resolution commutativity, then the P-inference is sound and complete.

An example of application of this result is to retrieve all the consequences about a given individual "a". Others are to retrieve the consequences related to a given topic of interest.

Fundamental Object Concepts and Constructions

Hans-Dieter Ehrich
Abteilung Datenbanken, Technische Universität, Postfach 3329
W-3300 Braunschweig, GERMANY

ehrich@idb.cs.tu-bs.de

(Joint work with Amilcar Sernadas)

Abstract

We provide a systematic framework where the concepts *object* and *class* and the constructs *inheritance* and *interaction* are clarified. Our object notion is based on that of a process, but the framework is independent of a particular process model. For illustration purposes, however, we outline two which emphasize the importance of process morphisms. There are categorial process models where limits reflect parallel composition and colimits reflect internal choice. Classes are shown to be special objects representing dynamic — and possibly polymorphic — collections of objects. Inheritance constructs are introduced as steps for building an inheritance schema: specialization, multiple specialization, abstraction, and generalization. Interaction constructs are introduced as steps for building an object community: incorporation, aggregation, interfacing, and synchronization. By using the same mathematics for both, a remarkable symmetry between inheritance and interaction constructs comes to light.

Visual Specifications of Conceptual Database Schemata

Gregor Engels
Leiden University, Dept. of Computer Science
P.O. Box 9512, NL-2300 RA Leiden, THE NETHERLANDS

engels@rulwi.LeidenUniv.nl

Abstract

Visual specifications are used in several disciplines, as, e.g., electrical engineering, and construction of houses. Their advantage is that they often have an intuitive semantics, which eases the understanding of those specifications. Visual descriptions are also used in different areas of computer science. Examples are control flow graphs, program dependency graphs, Entity-Relationship diagrams, Petri nets, or data flow diagrams.

The first part of the talk presents an approach to specify graphically the static structure of a system by extended Entity-Relationship diagrams. The local behaviour of objects is described by state transition diagrams. Some problems are discussed which occur if different views on the local behaviour of an object are combined within a complete state transition diagram.

In the second part, a graphical language is presented to specify complex actions on a set of objects. Complex actions are composed by the designer by using elementary actions. These descriptions are called "object flow graphs". Elementary actions are derived automatically from the description of the static structure of a system. They guarantee all inherent integrity constraints. This means that they do "update propagation", if this is necessary to yield a consistent global state after a local modification.

Within the CADDY environment, which has been developed at TU Braunschweig (Germany) during the last four years, tools have been realized to support the specification and interpretation of those complex actions. These tools are integrated within a set of tools which support the design and rapid prototyping of conceptual database schemata.

Process Semantics of Object Specifications

Jose Luiz Fiadeiro
Departmento de Matematica, IST
Av. Rovisco Pais, P-1096 Lisboa Codex, PORTUGAL

llf@inesc.pt

Abstract

A process semantics for temporal logic specification is provided by relating a category of temporal theories and interpretations between theories where specification configuration and interconnection is achieved via colimits of diagrams, and a category of algebraic models of processes where parallel composition is explained in terms of limits of diagrams. Given a diagram in the categories of theories and a model of it as a diagram in the category of processes, we prove that the limit of the process diagram is a model of the colimit of the theory diagram. That is to say, any denotation of a system of interconnected specifications corresponds to a configuration of their denotations as a system of interconnected processes.

Algebraic Semantics for the Object Paradigm

Joseph A. Goguen
Programming Research Group, University of Oxford
11 Keble Road, Oxford OX1 3QD, UK

Joseph.Goguen@prg.oxford.ac.uk

Abstract

Our goal is to extend the techniques of algebraic specification to the object paradigm in a way that integrates both the data and the process aspects, and that also handles concurrency, sharing, hiding and non-determinism in natural ways.

We first use object encapsulation to motivate some restrictions on order sorted signatures and morphisms, and then show that the result is an institution, in which satisfaction is behavioural satisfaction. Next, we show how theories and models in this institution formalise specifications and objects. Then, using the alternating bit protocol as an example, we show how sort constraints and a "wedge product" construction model forms of inheritance. We also use gluons, limits, and an op-Grothendieck flattening construction to handle the interconnection and behaviour of objects in complex systems. ("Gluons" are interface objects, limits give behaviour, and the op-Grothendieck construction allows limits of diagrams of models with varying signatures.)

We find that algebras of traces occur as initial models, and that arbitrary implementation algebras can be handled as well. We model processes with recursive equations on modadic operations.

Aspects of Inheritance in a Rule-Language

Georg Lausen and Heinz Uphoff
Fakultät für Mathematik und Informatik
Universität Mannheim, W-6800 Mannheim, GERMANY

{lausen | uphoff}@pi3.informatik.uni-mannheim.dbp.de

Abstract

The topic of the talk is the interaction of inheritance and deduction as it might occur in a rule-language. Problems arise if a rule becomes applicable due to inheritance of a certain value such that from the corresponding rule application it can be deduced that a different value should hold for the class from which inheritance took place. To be more specific consider the situation where there are three classes called *upper*, *middle* and *lower*; *lower* is a subclass of *middle* and *middle* is a subclass of *upper*. Now assume that a *method* *meth* gives the result a when applied on *upper* and further, that there exists a rule of the form $middle.meth \rightarrow b \Longleftarrow lower.meth \rightarrow a$ meaning that whenever *meth* applied on *lower* gives the result a, then the result of *meth* applied on *middle* is b. We can see that once we inherit $meth \rightarrow a$ from *upper* to *middle* and then from *middle* to *lower*, by application of the rule we get $middle.meth \rightarrow b$, which invalidates the inheritance from *upper* to *middle* and consequently from *middle* to *lower* and finally removes the applicability of the rule.

We present model-theoretic semantics which treats such cases in an intuitive way [1]; in the above example inheritance is only done from *upper* to *middle* - inheritance from *middle* to *lower* is blocked. We further discuss algorithms to compute the corresponding models. We implement the intuition of the suggested semantics by certain meta-rules and corresponding rewritings he original rules. Finally we discuss the relationship to well-founded semantics.

[1] M. Kifer, G. Lausen, J. Wu: *Logical Foundations of Object-Oriented and Frame-Based Languages.* Universität Mannheim, Reihe Informatik, 3/1990.

Semantics and Usage of Defaults in Specifications

Udo W. Lipeck
Institut für Informatik, Universität Hannover
Lange Laube 22, W-3000 Hannover 1, GERMANY

ul@informatik.uni-hannover.de

(Joint work with Stefan Brass, Miguel Dionisio (both Hannover)
and Mark Ryan (Imperial College London))

Abstract

The goal of this talk is to explain the application of hierarchical defaults, i.e. defaults with priorities, in logic-based specifications of information systems. We discuss the usefulness of defaults for different specification scenarios like specialization, aggregation, explanation, revision, etc. To understand defaults formally, we introduce a general framework which is parameterized on the underlying logical institution, provided that institutions are extended by an instantiation mechanism for formulae and by an appropriate choice of predefined interpretation parts. It is shown that hierarchical defaults have intended models if the extended institution is compact. As an example for a non-standard logic, we give the semantics of defaults in the multi-modal object calculus of the IS-CORE project. To structure and compose specifications with defaults, default-preserving specification morphisms are defined and corresponding colimit constructions are sketched.

Logical Aspects of Object–Oriented Systems Specification

Tom S. E. Maibaum
Department of Computing, Imperial College of Science
180 Queens Gate, GB-London SW7 2BZ, UK

tsem@doc.ic.ac.uk

Abstract

We bring together the use of temporal logic for specifying concurrent systems, in the tradition initiated by A.Pnueli, and the use of tools from category theory as a means for structuring specifications as combinations of theories in the style developed by R.Burstall and J.Goguen. As a result, we obtain a framework in which systems of interconnected components can be described by assembling the specifications of their components around a diagram, using theory morphisms to specify how the components interact. This view of temporal theories as specification units naturally brings modularity to the description and analysis of systems. Moreover, it becomes possible to import into the area of formal development of reactive systems the wide body of specification techniques that have been defined for structuring specifications independently of the underlying logic, and that have been applied with great success in the area of Abstract Data Types. Finally, as a discipline of design, we use the object-oriented paradigm according to which components keep private data and interact by sharing actions, with a view towards providing formal tools for the specification of concurrent objects.

Order Sorted, Hidden Sorted Refinement

Grant Malcolm and Joseph A. Goguen
Programming Research Group, University of Oxford
11 Keble Road, Oxford OX1 3QD, UK

{Grant.Malcolm | Joseph.Goguen}@prg.ox.ac.uk

Abstract

Order sorted specifications allow the description of algebras with partial operations; hidden sorted specifications introduce the notion of observational equivalence, as in the equivalence of black box automata. We present a definition of refinement for order- and hidden sorted specifications, and discuss a technique for proving that one specification is refined by another. Two examples of refinement are given, one of which suggests that the proof technique is particularly pertinent to object orientation.

A Meta-rule Approach to the Specification of Inference Methods in Rule-based Information Systems

Rainer Manthey
ECRC, Arabellastr. 17, W-8000 München 81, GERMANY

rainer@ecrc.de

(Joint work with François Bry)

Abstract

Rules, in the spirit of deductive database terminology, can be viewed as specifications of sets of data. They can even be considered executable specifications, as there are efficient standard procedures (such as differential fixpoint iteration) which materialize (i.e., compute and store) rule-defined data by successively applying rules to an initial database of stored facts. Meta-rules are rules specifying sets of meta-data.

The main message of this contribution is that meta-rules constitute a very elegant and powerful tool for specifying sophisticated inference methods. This is possible because inference processes are conveniently characterized by means of the meta-data they produce and manipulate. If performing inference in order to answer queries, one aims at reducing the search space by generating subqueries in a top-down manner before generating relevant intermediate results and answers bottom-up. Subqueries and answers constitute meta-data. The analogous observation holds for update-driven inference where the task is to determine the consequences of an update of a stored data set on the extension of rule-defined data sets. The motivation for doing so is the need etect changes of monitored conditions (such as integrity constraints or alerters). Again "blind" bottom-up propagation of updates can be avoided by prior top-down analysis of conditions to be monitored. Here induced updates and monitored conditions are the relevant meta-data. Both inference modes are very conveniently expressable using meta-rules. Applying fixpoint iteration to these meta-level specifications leads to efficient implementations, particularly if the need to access object-level rules is eliminated by partial evaluation. A well-known method for query-driven inference, the "magic sets" approach, can thus be easily explained by means of systematic partial evaluation of a meta-rule specification of query-driven inference. A similar approach can be obtained for update-driven inference as well.

Language Features for Object-Oriented Specification of Information Systems

Gunter Saake and Ralf Jungclaus
Abteilung Datenbanken, Technische Universität, Postfach 3329
W-3300 Braunschweig, GERMANY

{saake|jungclau}@idb.cs.tu-bs.de

(Joint work with T. Hartmann and C. Sernadas)

Abstract

In this talk we sketch the motivations for the development of the object-oriented conceptual modeling language TROLL and briefly introduce its basic features.

TROLL is particularly suited to be used in the early stages of information systems development. In those stages, we have to concentrate on abstract, implementation-independent descriptions of static and dynamic concepts that are relevant. Moreover, conceptual specifications must be formal because they act as "contracts" that are the basis for implementation. Finally, conceptual specification languages should offer a wide variety of language features that support structuring of specifications and "natural" (whatever this may mean) description of concepts (which also includes redundant language features).

The language TROLL tries to integrate object-oriented ideas with concepts of semantic data models and approaches to the specification of data types and concurrent systems. TROLL is based on sublanes for data terms, linear temporal logic and process specification. In TROLL, the description of static and dynamic aspects of objects is integrated in object descriptions.

The basic building blocks of TROLL specifications are *templates* that are descriptions of object prototypes. Object descriptions can be related in many ways. The abstraction mechanisms provided by TROLL are classification, specialization, roles, aggregation and interfacing. Abstraction mechanisms along with the basic structuring in object descriptions help in organizing the system specification.

In order to support the composition of system specifications from component descriptions TROLL provides language features to describe interactions and dependencies between components when put into the system context.

A complete language description can be found in [1].

[1] R. Jungclaus, G. Saake, T. Hartmann, C. Sernadas: *Object-Oriented Specification of Information Systems: The TROLL Language.* TU Braunschweig, Informatik-Bericht 91–04, 1991.

Object Template Institution

Amilcar Sernadas
INESC, Apartado 10 105, Rua Alvez Redol 9, 7a
P-1017 Lisboa Codex, PORTUGAL

acs@inesc.inesc.pt

(Joint work with J.F.Costa, J.L.Fiadeiro and H.-D.Ehrich)

Abstract

Given a category of object templates and morphisms between them, as well as an envisaged temporal logic of object template specification and verification, one faces the problem of setting-up the resulting institution if possible at all. A specific category of object templates (based upon sets of traces of sets of event symbols) and a specific logic (linear temporal propositional logic) are considered in order to illustrate the difficulties in the procedure of setting-up the institution. Basic requirements are: (1) the existence of a fibration from the category of templates to the category of alphabets of symbols; (2) the possibility of extracting a suitable Kripke structure from each template (in this case a multi-linear one); (3) the existence of a null event in the event-space of each template. If requirement (1) is fulfilled then the semantic functor of the institution appears as the pulling-back functor of the fibration. Requirement (3) is essential for the satisfaction condition (because of the special nature of the X temporal operator). Furthermore, one expects the category of templates to be complete (for parallel composition) and cocomplete (for choice). An additional requirement is needed to ensure that there is a canonical (terminal) template satisfying each object specification: it is sufficient to require that satisfaction is preserved by choice between templates over the same signature. This seems to be equivalent to ask for the existence of a satisfaction filtration. In these conditions, the colimit of a diagram of template specifications is satisfied by the limit (parallel composition) of a diagram of template models.

Graphical specification languages and specificational complexity

Arne Sølvberg
Dept. of Electrical Engineering and Computer Science
NTH, University of Trondheim, N-7034 Trondheim, NORWAY

asolvber@idt.unit.no

Abstract

Systems specifications serve the dual purpose of being the basis for

* exchanging systems knowledge among the system's designers, and

* creating an operational software system.

These two purposes may be conflicting, in particular when developing large systems. Because systems specifications must be both understandable and complete, the usual approach is to employ several sets of modelling constructs, which are fitted for the different purposes.

A proposal for a unified modelling language is presented. The language combines well-known systems development graphical specification methods e.g. ER-model, DFD-model, and extends those methods with modelling constructs which makes it possible to generate executable code. The automatic generation of industrial code, e.g. Ada, is shown to be possible, as well as the generation of (rapid) prototypes based on a combination of Prolog and C-code. It is argued that the proposal, which is mostly a graphical language, constitutes an example of an executable specification language.

When systems complexity increases one may experience some difficulties concerning the understandability of systems specifications in the proposed language. These problems may be solved by defining suitable abstractions of the detailed specifications. Specifications may then become enough simplified, so that they can be used as a basis for knowledge exchange among systems designers.

Functional Object-Oriented Design (FOOD)

T. H. Tse
Department of Computer Science
The University of Hong Kong, Pokfulam Rd, HONG KONG

tse@csd.hku.hk

(Joint work with Joseph Goguen)

Abstract

Object-oriented analysis and design methodologies are considered as the most popular software development methods for the 1990s. The informal graphical notations, mostly based on structured methodologies popular in the 1980s, are widely accept by practitioners. On the other hand, a number of formal object-oriented specification languages have been proposed, helping users to verify the correctness of the specification and implementation. They are, however, far from popular to systems designers in the industry.

Functional Object-Oriented Design (FOOD) is an attempt to provide a bridge between the popular object-oriented graphical notations and Functional Object-Oriented Programming System (FOOPS), which is a formal object-oriented programming language with formal algebraic semantics. We propose a set of graphical notations and methodology guidelines. The static relationships in a system, such as classes, data types, methods, attributes, functions, modules and inheritance, are specified in a notation based on data flow diagrams and enhanced entity-relationship diagrams. Behavioral properties of the system are defined by state-transition diagrams, multi-level data flow diagrams and object structure charts. All of these representations can be mapped directly into the corresponding declaration statements and axioms in FOOPS.

I LIKE MOKUM

Reind Van de Riet
Dept. of Mathematics and Computer Science
Vrije Universiteit Amsterdam (=Mokum)
De Boelelaan 1081 a, NL-1081 HV Amsterdam, THE NETHERLANDS

vdriet@cs.vu.nl

(LIKE is an acronym for Linguistic Instruments in Knowledge Engineering
MOKUM is an acronym for Manipulating Objects with Knowledge and Understanding in Mokum)

Abstract

In the process of constructing an information (=knowledge) system: (Requirements definitions- Analysis - Prototyping - Implementation) words are being used on all levels. Some words are being used as keywords, in which case their meaning is predetermined by the tools and structures used. Most words, however, are chosen by the information analysts, designers and programmers. They are used to denote entities, object-types, objects, attributes, values, etc. In this paper we will study the structure and use of a Lexicon in which these words are stored together with their meaning in an attempt to make reuse of specifications possible and to standardize the use of words. The meaning is laid down in a dictionary or Lexicon. It is shown how the Lexicon is constructed by taking a set of sentences describing in Natural Language, a certain Universe of Discourse and translating these sentences in CPL formulas. CPL is a formal language for specifying a Conceptual Model. It is also shown how from CPL programs can be generated which can be executed. Furthermore, it turns out that MOKUM was an appropriate language to define the Lexicon in.

The identification of objects and roles

Roel Wieringa and Wiebren de Jonge
Dept. of Mathematics and Computer Science, Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, THE NETHERLANDS

roelw@cs.vu.nl

Abstract

In order to identify real-world objects, you must classify them. An object may for example be one PERSON instance but two EMPLOYEE instances, if he or she has a job at two companies. This is because an EMPLOYEE is really a role of a PERSON and one PERSON can play several roles, even roles of the same type. What is called "class migration" is the phenomenon that one object starts or stops playing a role, such as that one PERSON starts or stops being an EMPLOYEE. Class migration is often represented by changing an identifier (as in the Orion system), but this destroys historical identity information. Class migration always involves making functions or predicates applicable during the life of an object that were not applicable before (i.e. delivered no value when applied to the object). Loss of historical identity information and partial functions or predicates can be avoided if roles get an identifier that, just like object identifiers, are globally unique and unchangeable. This leads to a new inheritance mechanism, from an instance to a role played by the instance. In this form of inheritance, a role delegates the answering of queries about attribute values to the instance (an object or another roles) that plays the role.

Persistance + Undoability = Transactions

Jeannette Wing
Lab. for Computer Science
545 Technology Square, Cambridge MA 02139, USA

jmw@lcs.mit.edu

Abstract

Persistence means objects live potentially forever. Undoability means that any change to a program's store can potentially be undone. In our design and implementation of support for single-threaded nested transactions in Standard ML of New Jersey (SML/NJ), we provide persistence and undoability as orthogonal features and combine them in a simple and elegant manner.

We provide support for persistence through an SML interface that lets users manipulate a set of persistent roots and provides a *save* function that causes all data reachable from the persistent roots to be moved into the persistent heap. We implement the interface through simple extensions to SML's generational garbage collector and maintain the persistent heap using CMU's Recoverable Virtual Memory system.

We provide support for undoability through an SML interface that exports two functions: *checkpoint*, which checkpoints the current store, and *restore*, which undoes all changes made to the previously checkpointed store. The implementation takes advantage of the simple runtime representation of data in SML and, as for persistence, extends the existing garbage collector scheme. SML's "mostly" functional nature allows us to implement this abstraction without undue performance penalty.

Finally, we combine these capabilities to support single-threaded nested transactions by defining a higher-order function *transact* that guarantees the permanence of effects of committed transactions. We succinctly define *transact* completely in terms of the interfaces for persistence and undoability. Unlike other transaction-based programming languages like Argus or Avalon/C++, we need not add new control structures; moreover, we handle aborts of nested or top-level transactions using SML's exception mechm.

A Sheaf Semantics for an Object-Oriented Language

David A. Wolfram
Programming Research Group, University of Oxford
11 Keble Road, Oxford OX1 3QD, UK

David.Wolfram@prg.oxford.ac.uk

(Joint work with Joseph A. Goguen)

Abstract

We give a sheaf semantics for most expressions of the object-oriented language FOOPS (*F*unctional and *O*bject-*O*riented *P*rogramming *S*ystem). Expressions can contain first-order functions, methods, and method combiners such as concurrent choice (||) or non-deterministic choice (or). There are built-in methods for deleting and creating objects. Evaluation of expressions can be done concurrently. Each function, attribute, method, and method combiner symbol has an associated evaluation strategy, or E-strategy, which specifies the order of evaluation of its arguments. The sheaves for functions and user-defined methods, built-in methods, method combiners, and the initial sheaf label the nodes of a directed graph called a diagram whose edges are labelled by natural transformations called sheaf morphisms. The semantics of expression evaluation is based on the limit of this diagram. This is a larger-scale example of a general theory of sheaf semantics.

Zuletzt erschienene und geplante Titel:

H. Alt , B. Chazelle, E. Welzl (editors):
Computational Geometry, Dagstuhl-Seminar-Report; 22, 07.10.-11.10.91 (9141)

F.J. Brandenburg , J. Berstel, D. Wotschke (editors):
Trends and Applications in Formal Language Theory, Dagstuhl-Seminar-Report; 23, 14.10.-18.10.91 (9142)

H. Comon , H. Ganzinger, C. Kirchner, H. Kirchner, J.-L. Lassez , G. Smolka (editors):
Theorem Proving and Logic Programming with Constraints, Dagstuhl-Seminar-Report; 24, 21.10.-25.10.91 (9143)

H. Noltemeier, T. Ottmann, D. Wood (editors):
Data Structures, Dagstuhl-Seminar-Report; 25, 4.11.-8.11.91 (9145)

A. Dress, M. Karpinski, M. Singer(editors):
Efficient Interpolation Algorithms, Dagstuhl-Seminar-Report; 26, 2.-6.12.91 (9149)

B. Buchberger, J. Davenport, F. Schwarz (editors):
Algorithms of Computeralgebra, Dagstuhl-Seminar-Report; 27, 16..-20.12.91 (9151)

K. Compton, J.E. Pin , W. Thomas (editors):
Automata Theory: Infinite Computations, Dagstuhl-Seminar-Report; 28, 6.-10.1.92 (9202)

H. Langmaack, E. Neuhold, M. Paul (editors):
Software Construction - Foundation and Application, Dagstuhl-Seminar-Report; 29, 13..-17.1.92 (9203)

K. Ambos-Spies, S. Homer, U. Schöning (editors):
Structure and Complexity Theory, Dagstuhl-Seminar-Report; 30, 3.-7.02.92 (9206)

B. Booß, W. Coy, J.-M. Pflüger (editors):
Limits of Modelling with Programmed Machines, Dagstuhl-Seminar-Report; 31, 10.-14.2.92 (9207)

K. Compton, J.E. Pin , W. Thomas (editors):
Automata Theory: Infinite Computations, Dagstuhl-Seminar-Report; 28, 6.-10.1.92 (9202)

H. Langmaack, E. Neuhold, M. Paul (editors):
Software Construction - Foundation and Application, Dagstuhl-Seminar-Report; 29, 13.-17.1.92 (9203)

K. Ambos-Spies, S. Homer, U. Schöning (editors):
Structure and Complexity Theory, Dagstuhl-Seminar-Report; 30, 3.-7.2.92 (9206)

B. Booß, W. Coy, J.-M. Pflüger (editors):
Limits of Modelling with Programmed Machines, Dagstuhl-Seminar-Report; 31, 10.-14.2.92 (9207)

N. Habermann, W.F. Tichy (editors):
Future Directions in Software Engineering, Dagstuhl-Seminar-Report; 32; 17.2.-21.2.92 (9208)

R. Cole, E.W. Mayr, F. Meyer auf der Heide (editors):
Parallel and Distributed Algorithms; Dagstuhl-Seminar-Report; 33; 2.3.-6.3.92 (9210)

P. Klint, T. Reps (Madison, Wisconsin), G. Snelting (editors):
Programming Environments; Dagstuhl-Seminar-Report; 34; 9.3.-13.3.92 (9211)

H.-D. Ehrich, J.A. Goguen, A. Sernadas (editors):
Foundations of Information Systems Specification and Design; Dagstuhl-Seminar-Report; 35; 16.3.-19.3.9 (9212)

W. Damm, Ch. Hankin, J. Hughes (editors):
Functional Languages:
Compiler Technology and Parallelism; Dagstuhl-Seminar-Report; 36; 23.3.-27.3.92 (9213)

Th. Beth, W. Diffie, G.J. Simmons (editors):
System Security; Dagstuhl-Seminar-Report; 37; 30.3.-3.4.92 (9214)

C.A. Ellis, M. Jarke (editors):
Distributed Cooperation in Integrated Information Systems; Dagstuhl-Seminar-Report; 38; 6.4.-8.4.92 (9215)