IMPRESS Database Design Tool - a high-level design toolset based on formal theory

J. Flokstra, M. van Keulen and J. Skowronek
Informatics Faculty, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
e-mail: {flokstra, keulen, skow}@cs.utwente.nl

Abstract

This document presents the Database Design Tool prototype, developed at the University of Twente. The Tool is used to specify databases in a graphical way, and is based on a formal specification language TM (described in ECOOP'93 article [2]). TM and the Database Design Tool support object-oriented concepts such as classes, object, methods and inheritance. The main point we want to state is that software engineering based on a sound formal basis does not have to sacrifice ease-of -use and flexibility; we state that, on the contrary, it is this formal basis which proves to be beneficial and profitable for the user - enabling faster and error-free software development.

I. FORMAL BASIS OF SOFTWARE ENGINEERING

In recent years, there has been a growing interest in the area of formal founding of the software engineering process. The sizes of applications, the complexity of the defined schemas, the variety and complexity of planned actions have proved to be prohibiting for traditional and intuitive methods of software engineering. It is this complexity which makes it more and more difficult for humans to perform all of the software engineering activities. That is why there is an urgent need for tools which can enable modeling and design on much a higher level, supporting automatic translation into low-level design or program. However, high-level design and modeling will be accepted and used in real-world situations, if it will provide (among others) two important (but often perceived as mutually exclusive) features:

- easy-to-use, intuitive design tools
- foundation on a formal theory, enabling various type- and consistency checks

In the software engineering domain and in the industry, there is a wide-spread belief that modeling and design based on a formal basis only introduces unneeded complexity and does not deliver the tools the designers need. We feel, however, that the complexity of current applications makes the use of formal techniques a necessity, and it is the task of the tools designers to provide intuitive user interfaces, so that the formal theory underlying the tools becomes a benefit rather than a hindrance. Our tools attempt to provide just the functionality: giving the designer the powerful apparatus of formal set theory and object-oriented concepts, in the same time, however, providing well-known graphical interface.

On the other side, in the research world there is a certain reluctancy to provide the real-world users with the interfaces and tools they need, rather being satisfied with the development of the underlying theory. We feel that providing those layers is profitable for both sides: there is an increased recognition of research efforts, and the real-world users get the benefit of formality and additional features. The IMPRESS database Design Tool was created with above assumptions in mind.

II. PROTOTYPE DESCRIPTION

The demonstration will present the Database Design Tool prototype developed in the IMPRESS project (Esprit project 6355). The IMPRESS project started in May 1992 and aims at creating a low-level storage manager tailored for multimedia applications, together with a library of efficient operators, a programming environment, high-level design tools and methodology. The DDT is part of this last effort. It is based on the specification language TM described in [2], [3].

A. Goals

The goal of the DDT is derived from the application requirements defined by the partners in the project (including industrial partners meant as the first users).

In the area of Technical Information Systems, there is a need for tools supporting modeling of complex objects. Designers start usually with a partial or incomplete model that is further refined when new attributes and new relations are discovered or when knowledge about the object evolves. This is referred to as (desirably planned) incremental design or step by step prototyping. The process seems to be well suited for users coping with uncertainty about their own needs or requirements and allows great flexibility for evolution. This design process applies to database

schemas as well as access programs. The IMPRESS DDT is aimed at supporting this process.

In the context of modern Technical Information Systems, we observe that applications' size and complexity is of such scale that using an underlying formal model with its supporting tools seems to be the only way of ensuring consistency and correctness. We thus propose basing the design process on TM, a database specifi cation language, which allows performing consistency checks on the specifi cations, as well as constructing semi-automatically formal proofs of correctness. The IMPRESS DDT supports specifi cation of database schemas (hierarchies of classes), the attributes defi ned on the schema as well as methods and constraints, specifi ed in high-level, declarative method part of the language.

Basing the tools on a formal specifi cation language must not preclude providing easy-to-use, graphical interfaces. The DDT provides graphical tools for:

- creation of the database schema using object-oriented concepts such as classes and inheritance, specifying constraints and methods on the classes defi ned in the schema (design),
- instantiation of the objects in the schema, as well as validation of the schema and operations by executing them on a prototype (prototyping)
- various checks on the specifi cation, such as type checking, safeness detection, constraint analysis, documentation, and translation to executable language

The DDT contains a graphical editor for the database schema. It provides facilities to easily (graphically) defi ne database entity types, such as classes, attributes, types, and inheritance structure. Constraints and methods can be specifi ed in TM using a syntax-directed editor. The DDT contains features for specifying application classes and methods, some of them using database classes and methods. The DDT contains a generator for rapid prototypes which can be used for testing facilities (perform updates and queries on a test database, as well as schema modifi cation and simulation experiments). Such prototyping will enable early detection of specifi cation faults, on a small database, before the specifi cation is committed for realization.

B. Functionality

Functionality of the DDT was derived from the goals and requirements. It encompasses:

- fl exible, graphical design of modular database schemas
- handling complex objects in an effi cient manner
- specifi cation of constraints on various levels of granularity using high-level declarative languages
- specifi cation of methods using high-level declarative language
- compile-time type checking of the schema, constraints and methods
- schema modifi cation and simulation experiments in prototype versions of the database
- constraint analysis consisting of semi-automatic verifi cation of correctness-preserving properties after invocation of database transactions
- documentation and ordering facilities, providing fl exible "annotation"-like facilities for specifi cation

The DDT consists of (see further for introduction to TM):

- Graphical Interface (GI), which is used to design the database schema together with methods and constraints, as well as application classes and methods using database classes and methods
- TM tools, which include the prototyping environment (PTE), safeness detector and other tools

The GI provides the following functionality:

- graphical display/manipulation of database classes, attributes of those classes, inheritance hierarchies
- display/editing of constraints and methods defi ned on classes
- display/editing of application classes and methods

The PTE provides following functionality:

- population of the database with objects based on the schema created in the GTI
- evaluation of database methods and constraints on instantiated objects
- viewing the complex objects being results of the above evaluations, using the point-and-click interface to navigate through the complex object, thus enabling "debugging" of the specifi ed methods

All those functions are supported by an intuitive graphical interface, using windows- and mouse-based interaction.

C. TM and its role in the DDT

The DDT uses the formal language TM as its kernel. TM is a high-level language for the design and specifi cation of object-oriented database schemas in an effi cient and effective manner The TM language and its accompanying design tools enable users to perform complex semantic analyses of schemas, thus paving the way to a complete debugging of the conceptual design. As a design language, TM is equipped with powerful structuring primitives which enable a user to arrive at natural and intuitively correct designs. These structuring primitives are characterized by the following features

- Encapsulation (The concepts of Module and Class)
- Multiple inheritance

- Object-oriented specialization (Objects are not only specialized by adding attributes; already existing attributes can also be subject of specialization)
- Complex objects (Records, lists, sets, variants; and all arbitrarily nested)
- Methods and method inheritance
- Static constraints of different granularity (Object, Class-, and Database level, described by a full first-order typed logic)
- Composition links (Direct references to other objects as values of attributes)
- Static type checkability (The language has a complete formal basis)

We note that the TM language has a complete formal semantics ([2]), and it is this property of having a formal semantics that actually creates the possibility of having an integrated tool set, as TM does. Having a formal semantics entails that all expressions in the language have a precise and unique meaning; without such a non-ambiguous meaning for all constructions occurring in the language, it is impossible to build a reliable toolset supporting the language features involved. A designer using TM does not necessarily have to have knowledge of TM's underlying formal basis to achieve correct specifications of TM schemas, but it should be a reassuring fact that the TM toolset is for a large part the result of careful research depending on TM's well-established mathematical semantics.

The TM language is a specification language that is formally founded in the language FM. FM, in turn, is a language that is based on the ideas of Luca Cardelli. It can be seen as a strongly typed lambda calculus that allows for subtyping and multiple inheritance. Over the past four years the theory of FM has been developed by Balsters, Fokkinga, and de Vreeze [4] to exploit the ideas of Cardelli and to augment the theory to make it one that is suitable for object-oriented database specification.

The language TM should be understood as a syntactically sugared version of FM. First ideas on this language originated in discussions during a research stay in Milano in September 1990 [2]. This is why the language is called TM: it stands for Twente-Milano. Most of the present syntactical constructs of the language were developed in 1991. A redesign of the language took place in the first half of 1993.

D. Modeling of databases and applications using DDT and TM

In the IMPRESS project, we develop a comprehensive methodology to accompany the tools, even though we claim that the tools do not force the use of any specific methodology (rather they provide the formal means of validating designs, which in turn can be created in a different way).This methodology is based on a comparison of existing methodologies and a study of their suitability for Technical Information Systems.

In IMPRESS, we see the *Database* as the persistent data and associated methods, which can be used by different applications, each of which contains the following parts:

- *Model View*: containing all data (transient or persistent) and functions used in one application: persistent data may come from the Database
- *Man-Machine Interface*, responsible for communication and interaction with the end-users
- *Translator*, which synchronizes MMI events with Model View functions

In this view of the application, the DDT is used to design the Database as well as the Model View. Both of those parts are designed using the Graphical Interface. The design of the Database classes and methods is in this case followed by the design of application classes and methods. As both activities are performed in the same tool using the same graphical symbols, the cooperation of database designer and application designer (not necessarily the same person) is optimal. The approach outlined enables reuse of Database part in different Model View components, suitable for different application needs. Often, the Model View component will just extract the classes defined in the Database and will define additional classes and components.

E. Implementation state

Currently (April 1994) we are presenting the prototype of the Graphical TM Interface and the Prototyping Environment. Both tools make use of the TM Type Checker and TM-to-SPOKE translator. SPOKE is the language used in the execution of TM constraints and methods. TM as a high-level formal specification language is thus translated into the programming language SPOKE. Both GTI and PE are also written in SPOKE, using Motif graphical interface. The translators and the type-checker are written in C++.

F. Future plans

In the course of the project, other parts of the DDT are being implemented:

- Safeness Detector, which analyses a TM specification, detecting non-constructive expressions (evaluating to infinite sets); prototype implemented using LIFE
- Constraint Analyser, which examines the constraints and methods in the specification and tries to prove that certain update methods leave certain constraints invariant (meaning that these con-

straints need not to be evaluated after the execution of the method).

- Instance Editor, which enables creation of database objects in a graphical way, using the TM data model
- Documentation Facilities, built on WEB language, which enable the designer to combine natural language text with formal specifi cation text. Such a document then serves as a database specifi cation as well as a documentation of that specifi cation.

III. PUBLICATIONS

[1] R. Bal & H. Balsters, " A Deductive and Typed Object-Oriented Language", in Third International Conference on Deductive and Object-Oriented Databases, December 6-8, 1993, Scottsdale, Arizona, USA.

[2] H. Balsters, R. A. de By & R. Zicari, " Typed sets as a basis for object-oriented database schemas", in Proceedings Seventh European Conference on Object-Oriented Programming (ECOOP), July 26-30, 1993, Kaiserslautern, Germany, 1993.

[3] H. Balsters, R. A. de By & C.C. de Vreeze, " The TM manual", University of Twente, technical report INF 92-81, Enschede, 1992.

[4] H. Balsters & M.F. Fokkinga, " Subtyping can have simple semantics", Theoretical Computer Science 87 (September, 1991), pp. 81-96.

[5] H. Balsters & C.C. de Vreeze, " A semantics of object-oriented sets", in The Third International Workshop on Database Programming Languages: Bulk Types & Persistent Data (DBPL-3), August 27-30, 1991, Nafplion, Greece, P. Kanellakis & J.W. Schmidt, eds., Morgan Kaufman Publishers, San Mateo, CA, 1991, pp. 201-217.

[6] C.C. de Vreeze, " Extending the Semantics of Subtyping, accommodating Database Maintenance Operations", University of Twente, Enschede, The Netherlands, August 1989, Doctoraal verslag.

[7] C.C. de Vreeze, " Formalization of inheritance of methods in an object-oriented data model", University of Twente, Technical Report, INF 90-76, Enschede, December, 1990.

IV. SOFTWARE & HARDWARE REQUIREMENTS

Hardware:

Sun Sparc with color monitor
16 MB disk space minimum
16 MB memory minimum

Software:

X Windows R5 or OpenWindows
SunOS 4.1.1. or later