

Object-Level Modeling and Testing – Daskalos Experience

Jiří Brožek¹, Vojtěch Merunka^{1,2}, Tomáš Richta², and Václav Vostrovský¹

(1) Department of Information Engineering, Faculty of Economics and Management
Czech University of Life Sciences Prague

Kamýčká 129, 165 21 Prague 6 - Suchbátka, Czech Republic

{merunka,brozek,vostrovsky}@pef.czu.cz

(2) Software Engineering Research Group, Department of Computer Science
Faculty of Electrical Engineering, Czech Technical University in Prague

Karlovo nám. 13, 121 35 Prague 2, Czech Republic

{richtt1,merunka}@fel.cvut.cz

Abstract. The paper deals with the new paradigm of object-level modeling and testing. First part of the paper describes the method itself and its role in the area of software engineering. Second part presents our approach integrated with object database technology and lambda-calculus theory. Next, a supporting tool we developed is discussed. This part depicts our experience. Object-level modeling and testing is based on manipulation with particular object instances containing real data in similar way as in database querying. This enables to validate, verify and refine the conceptual model even before its final software implementation. The idea of object-level modeling is already being implemented in some educational or development tools (e.g. BlueJ and Visual Studio .NET). However these tools are too much oriented to target programming environment (Java or C#) and are focused on class-level models of software implementation. This is why we decided to implement our own tool Daskalos, which combines graphical user interface with lambda-expressions. Daskalos is developed in Smalltalk programming language and its user interface is inspired by the Self programming environment and Gemstone object database. The method and tool have been used in formal design and software engineering education at the University of Thessaly in Volos, Lehigh University in Pennsylvania, Czech University of Technology and Czech University of Life Sciences Prague. It is also used for rapid prototyping in software development projects. Interesting outcomes of our experience are not only related to application programming, but also to pure object database technology and business engineering, where instance-level models refines the business process design.

Keywords: Formal methods education, UML, object-level modeling, object-level testing, conceptual modeling, lambda-calculus, MDA, OCL, CASE tool, OOP, object-oriented databases, Smalltalk, Self

1 Introduction

Usual approach to software development considers testing as a part of project's implementation phase. As many modern methodologies (eg. MDA) start from abstract, human elaborated specifications, it is necessary to test the emerging design right in its conceptual phase long way before the project is implemented. This allows us to validate, verify and refine the conceptual model and thus greatly reduces the need for further changes in the implementation phase.

Like Haworth et al. has mentioned object-oriented software development is centered on class-based definitions. Objects are produced by instantiation of classes. Inside objects, methods interact with other objects by invoking its methods. Methods could be also dependent on previously executed methods. The interactions made by methods make connections among objects, which should be verified and tested [1]. It is where object level testing takes its place.

The paper presents our approach to object-level testing and also introduces this method as a part of software engineering curriculum at many universities. This goes hand in hand with our endeavor to use "objects first" approach in education [3]. (Objects-first approach is aimed on students of programming and introduces the concept of object-oriented paradigm and development using manipulation with object components.)

2 Object-Level Modeling and Testing

Object-level testing is the approach to start modeling the system as a set of example objects, that are independent and behave like real objects, contain data and are able to respond to messages. Those objects make a network connected by object dependencies. Such a network of objects represents the simulation of future software and thus could serve as verifying and validating platform for software development [1],[15].

This method could also well serve as an education platform for the objects-first approach in programming. Students are allowed to model the situation or software as a set of interconnected objects and immediately see the results of their work without need to build any window application for these objects.

3 Tools for Object-Level Modeling and Testing

3.1 BlueJ

Michael Kölling primarily developed BlueJ learning environment in late 1990s as the reaction to then unsatisfactory status of development environments used for teaching programming languages. It was called Blue system and consists of the special object-oriented language developed for teaching. This environment was later transferred to Java and called BlueJ. The project is now maintained by Deakin University, Melbourne, Australia, and the University of Kent in Canterbury, UK [16].

BlueJ visualizes the class structure and also allows visually modifying the class definition. After this the environment itself generates the source codes. BlueJ also allows to instantiate the classes and sends the methods to newly created objects to verify the designed model.

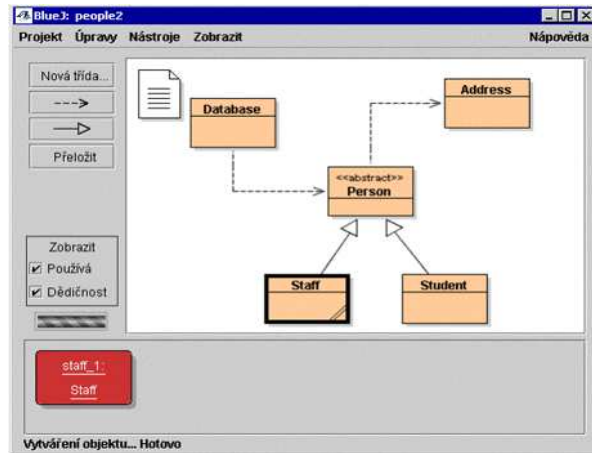


Fig. 1. BlueJ

While BlueJ was primarily intended for teaching the principles of object oriented programming in Java, it could also serve as a platform for verification and validation of the analytical model in the early phases of software development process [4],[5].

3.2 .NET Object Test Bench

Inspired by the success of the BlueJ, Microsoft implemented the Object Test Bench into their Visual Studio .NET. This tool also enables the user to create objects based on the class definition and execute designed methods and work with their data [17].

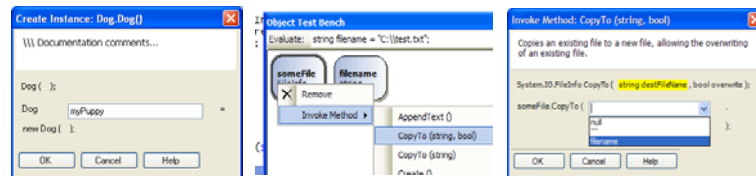


Fig. 2. Object Test Bench

However both of these environments are based on the approach class diagram first. This requires a higher level of the abstract way of thinking. That's the main difference from Daskalos, which uses the objects-first approach. We believe based on our experience that this is more natural way of modeling. Users create objects and connect them visually. Class diagram is then made automatically as a result of created objects properties.

4 Our approach: Data Modeling with Lambda-Calculus

More formally, our approach is based on the lambda-calculus adopted for the object-oriented approach.

The lambda-calculus is a surprisingly simple yet powerful system. It is based on function abstraction, to generalize expressions through the introduction of names, and function application, to evaluate generalized expressions by giving names particular values. The lambda-calculus has a number of properties, which suit it well for describing programming languages. First of all, abstraction and application concepts are all that are needed to develop representations for arbitrary programming language constructs. Thus, lambda-calculus can be treated as a universal machine code for programming languages. Furthermore, there are well-developed proof techniques for the lambda-calculus and these can be applied to descriptions of other programming languages. Finally, because the lambda calculus is very simple it is relatively easy to implement, as it is visible in our version of the Smalltalk programming language, for example [13],[14].

4.1 Object definitions with lambda-calculus

In our approach, we use standard non-typed form of the lambda-calculus for describing object behavior. This behavior consists of object methods and operations with objects. Complex definition of various calculi for objects is described in [12]. For better use of this simple form of calculus, we introduced four new concepts:

1. Operator \triangleleft , which expresses the message send to an object.
2. Variable σ , which refers to an object, who is the owner of the context, where this concept is present. (It is formal abstraction of variables `this` or `self`, well known from object-oriented programming languages.)
3. Operator $//$, which denotes selection from some set of objects.
4. Operator \gg , which denotes projection or another similar transformations of some set of objects.

For example, if we have a concrete object *customerX* of class *Customer*, which can be written as

$$\text{class}(\text{customerX}) = \text{Customer},$$

then we can write messages which access some real data from this object like

$customerX \triangleleft name \Rightarrow \text{John.}$
 $customerX \triangleleft surname \Rightarrow \text{Green.}$
 $customerX \triangleleft birthdate \Rightarrow 6/20/1968.$
 $customerX \triangleleft address \Rightarrow \text{Boston.}$

In our level of abstraction, these message sends are the same as accessing attributes of objects. Moreover, our formalism also enables to describe methods, which stores new data to objects as for example

$customerX \triangleleft address: \text{Philadelphia.}$

Methods themselves can be described as well. Each method can be interpreted as a lambda-expression (method body) associated with a name (method header). Let imagine the method *age*, which computes value of age of any person from their birth date as

$\langle age, (today - \sigma \triangleleft birthdate) / 365.2422 \rangle.$

If this method belongs to the set of methods of class *Customer* from previous example as

$\langle age, (today - \sigma \triangleleft birthdate) / 365.2422 \rangle \in methods(Customer),$

then we can access age in the same way as another attributes of the object *customerX* as

$customerX \triangleleft age \Rightarrow 39.$

4.2 Manipulation with objects using lambda-calculus

Object-level testing is based on the work with real object instances. We prefer the style analogous to database querying. There is analogy in selection, projection and other collection operations, but in contrast to the standard database technology, there is no strong dependence on concrete programming environment, storage problems, client-server deployment etc. We believe, that manipulations with collections having objects are the best way how to validate and verify the design of these objects [6],[7].

Lambda-calculus we introduced is capable to express these manipulations. For example, if we have set of contracts named *Contracts*, we can write selection for contracts on price bigger that 50\$ as follows:

$Contracts // (\lambda x | x \triangleleft price > 50).$

Lambda-expression in this example defines the selection rule. Another example is projection of the set of contracts to the set of products within these contracts. We assume that there is attribute product at each contract:

$Contracts \gg (\lambda x | x \triangleleft product).$

Lambda-expression in this example defines the transformation of each contract to the product ordered by this particular contract.

5 Daskalos

Daskalos is a computer program, which is used for the matter of object-oriented modeling in accordance to what is discussed in this article. It is programmed as a separate parcel of the system VisualWorks/Smalltalk version 7.4.1. System VisualWorks/Smalltalk is free for education and research purposes [18].

5.1 Smalltalk as the appropriate programming language for OOP theory

Smalltalk was developed in California in Palo Alto Research Centre (PARC) by a team of guiding scientists, Dr. Alan Kay (team of Learning Research Group) and Dr. Adele Goldberg (team System Concepts Laboratory) in the years 1970-1980. Subject of the whole research, which was remarkably financed by Xerox, was the project *Dynabook* for the evolution of the future personal computers.

The computer *Dynabook* was composed by bitmap graphical display, with new peripherals pen, later on mouse and tablet. Among its components had to be network interface as well. For the user-interface was for the first time in the world used overlapping windows, pop-up menu and icons. During the 1970s were actually manufactured some prototypes of such computers. It was assumed that computer will include standardized software environment, which will play together the role of an operating system and of a programming language with developing tools in one integrated environment. The name that was given to this software is *Smalltalk*.

Smalltalk was the almost completed project in the end of the year 1980. Mostly used concepts for it were from the language LISP, Logo and from the first object-oriented language Simula. Part of the development team remained in PARC and was supervised by A. Goldberg's company ParcPlace Systems (now fused with Cincom), which develops Smalltalk till today. Others together with A. Kay moved to Apple Computers, where they afterwards introduced in the market the first popular personal computer Lisa with graphical user interface. Smalltalk and its graphical user interface were subsequently used in the UNIX workstations. The first one was Tektronix 4404 in 1982. Contemporary Smalltalk is mainly used in the USA. Nowadays there are for instance large enterprise information systems of Navigant International Northwest Travel, Florida Power and Light, Orient Overseas Container Line or JP Morgan. Smalltalk is popular a a research tool as well. The idea of the Smalltalk GUI inspired the concept of the systems Macintosh OS, MS Windows, X-Window etc. Smalltalk directly influenced the concept of many programming languages and the standard of UML. Smalltalk is the very used system language in object-oriented databases.

Except for the commercially worldwide used Cincom VisualWorks, there are also two high-quality freeware implementations: Smalltalk/X and Squeak. Smalltalk-X is interesting for its embedding of the language C and is available on many computers with UNIX. Squeak is very popular and supported in the north-American universities. It continues the original idea of the Dynabook.

It is also popular as an experimental platform for research of innovative user-interfaces and operation systems (eg. project Croquet having 3D user interface and behaving as an network operation system).

5.2 Language examples

Smalltalk is integrated with its programming environment. Everything is in source code (and this happens also in commercial implementations). The system acts like the unique large-scale collection of living objects.

Important parts of the Smalltalk language are blocks of expressions. Blocks are straightforward implementation of lambda-expressions. A block is an independent object, can be assigned to a variable, several messages can be sent to it, and can be used in other expressions (messages) as a parameter. According to similar principles object methods are implemented as well. The following example shows a block, which is stored to an object with the name *B*. The block includes code, which raises the input value to square and adds the value of the object *A*:

formally $B \Leftarrow (\lambda x \mid x^2 + A).$
Smalltalk `B := [:x | (x ** 2) + A].`

This block can be then used as example:

```
A := 10.  
B value: 3.                   gives us the result 19 (32 + 10)  
                                  or  
A := 5.  
B value: 4.                   gives us the value 21 (42 + 5).
```

Other interesting feature of the Smalltalk language is the message *perform*. It is a message, which sends another entire method to the receiving object. For example an expression

```
M := #factorial.  
5 perform: M.                which returns result 120,
```

is equal to invoking method *factorial* on object 5 as `5 <factorial`.

Thanks to the library of Smalltalk classes it is possible to use Smalltalk as query language for object databases. This is shown in the following example:

formally $[2, 3, 6, 5, 4, 7, 8] \parallel (\lambda x \mid x > 5).$
Smalltalk `#[2,3,6,5,4,7,8] select: [:x | x > 5].`

which returns result `#[6,7,8]`.

5.3 Daskalos Application

We developed Daskalos as an application encapsulating original Smalltalk programming environment. In Daskalos it is possible to create classes, solitaire object instances and collections of objects in visual manner. All objects are displayed in UML standard. Methods and other expressions can be written in simplified form of Smalltalk language. This subset of Smalltalk has the same functionality as the language OCL (part of UML as well). Moreover, syntax of our language is very similar to syntax of OCL.

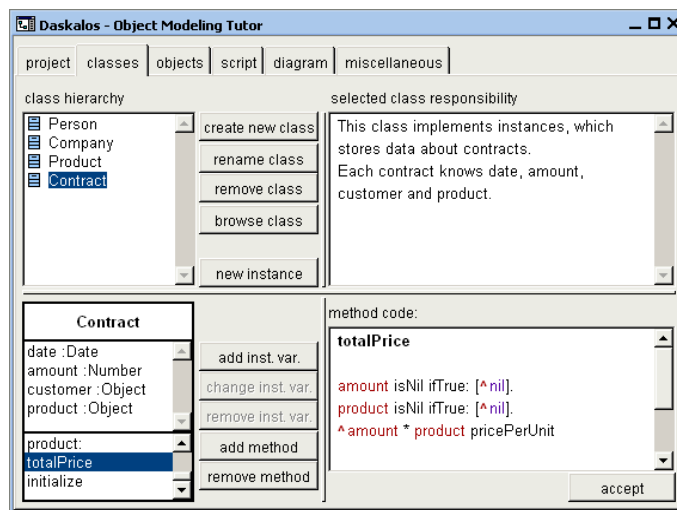


Fig. 3. Classes and methods

Daskalos allows working with particular objects in separate windows. This feature is inspired by the Self system [2]: Mouse clicks invokes methods, data can be manipulated using drag and drop (Fig.4).

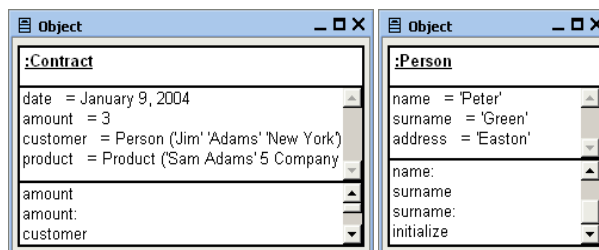


Fig. 4. Particular object instances in independent windows

There is a working panel designed for more complicated manipulation with objects like setting queries for object collections, for example (Fig.5 and 6).

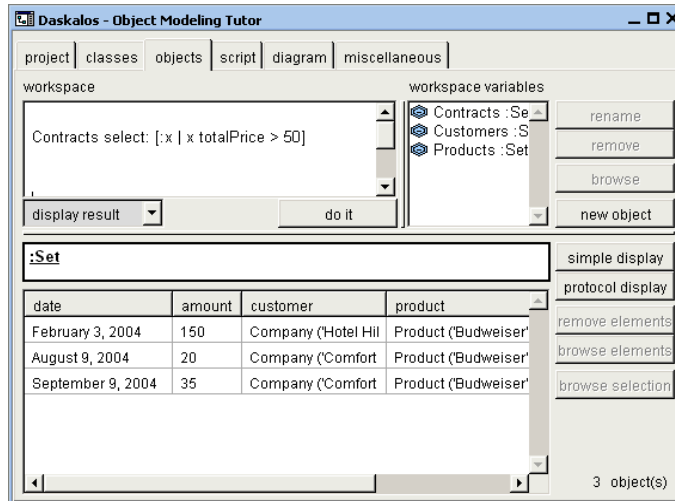


Fig. 5. Expression $Contracts // (\lambda x | x \triangleleft price > 50)$

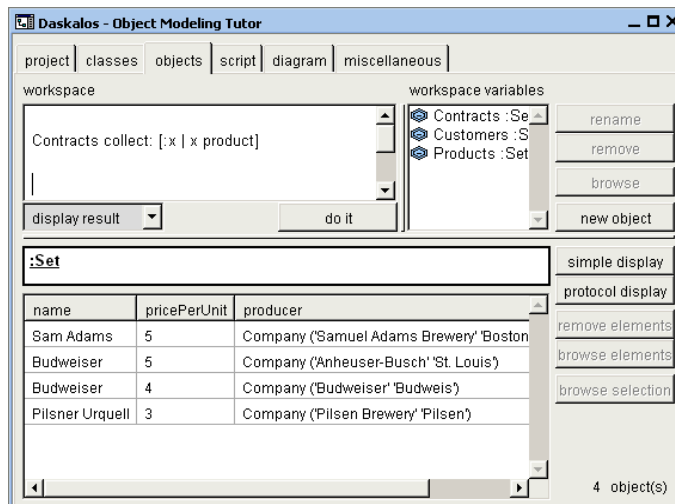


Fig. 6. Expression $Contracts \gg (\lambda x | x \triangleleft product)$

In Daskalos, UML class diagram is generated automatically from the database of tested objects, which must exist. (This is Daskalos' unique feature in comparison to other systems.) This means that analysts should start with particular objects having real data and messages. Generated UML diagram then concurrently shows the actual stage of this analysis process (Fig.7).

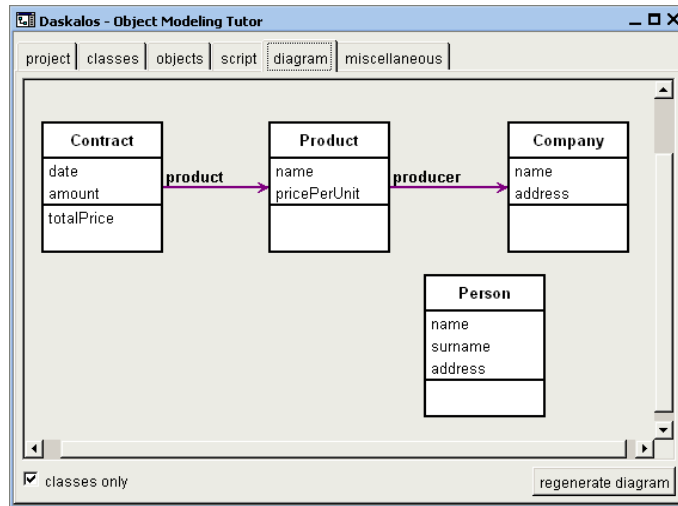


Fig. 7. Panel with diagram

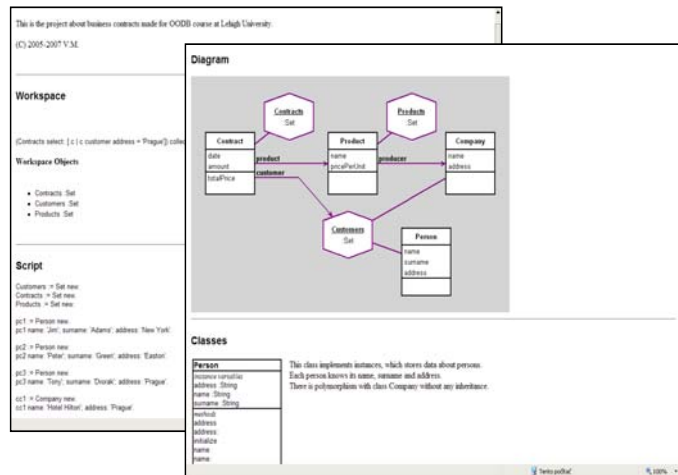


Fig. 8. HTML documentation

Daskalos is primarily designed for our OOP courses. For this reason, Daskalos saves HTML documentation typically used for student semester projects (Fig.8).

6 Conclusion

The method and tool have been used in courses of formal design and software engineering at Lehigh University in Pennsylvania, University of Thessaly in Volos, Czech Technical University and Czech University of Life Sciences. It is also used for rapid prototyping in software development projects which we take part of.

Interesting outcomes of our experience are not only related to application programming, but also to pure object database technology. Daskalos concept of object collections is fully compatible with the Gemstone object database environment [10].

Another experience is from the area of business systems modeling. We teach information management courses where business process models are created. These business processes are subsequently refined into particular actions, which should be tested. Daskalos querying is used here, because standard business modeling CASE tools do not provide any support for design testing.

The authors would like to acknowledge the support of the research grant project MSM6046070904 of the Czech Ministry of Education.

References

1. Haworth B., Kirsopp C., Roper M., Shepperd M., Webster S.: *Towards the Development of Adequacy Criteria for Object-Oriented Systems* (1997)
2. Self – [http://en.wikipedia.org/wiki/Self_\(programming_language\)](http://en.wikipedia.org/wiki/Self_(programming_language))
3. Meyer B.: *Towards an Object-Oriented Curriculum*, Object Currents – www.sigs.com/publications/docs/oc/9602/oc9602.c.meyer.html
4. Conference proceedings of ECOOP, *European Conference on OOP Programming*
5. Conference proceedings of OOPSLA, *Conference on Object-Oriented Programming Systems, Languages and Applications*
6. Ambler S.: *Building Object Applications That Work, Your Step-By-Step Handbook for Developing Robust Systems Using Object Technology*, Cambridge University Press/SIGS Books, 1997, ISBN 0521-64826-2
7. Ambler S.: *Object Orientation – Bringing data professionals and application developers together*, <http://www.agiledata.org/essays/objectOrientation101.html>
8. Barry D.: *The Object Database Handbook: How to Select, Implement and Use Object-Oriented Databases*, ISBN 0471147184
9. Blaha M., Premerlani M.: *Object-Oriented Modeling and Design for Database Applications*, Prentice Hall, ISBN 0-13-123829-9
10. Gemstone Object Server – documentation & non-commercial version download, <http://www.gemstone.com>, <http://www.smalltalk.gemstone.com>
11. Yonghui Wu, Zhou Aoying: *Research on Normalization Design for Complex Object Schemes*, Info-Tech and Info-Net, vol 5. 101-106, Proceedings of ICII 2001, Beijing
12. Abadi M., Cardelli L.: *A Theory of Objects*, Springer, 1996, ISBN 0387947752

13. Michaelson G.: *An Introduction to Functional Programming Through Lambda Calculus*, Addison-Wesley, 1989, ISBN 0201178125
14. Barendregt H., Barendsen E.: *Introduction to Lambda Calculus*. Technical report, Department of Computer Science, Catholic University of Nijmegen, July 1991
15. Liu L., Roussev B.: *Management of the Object-oriented Development Process*, Idea Group Publishing, 2005, ISBN 1591406048
16. BlueJ Project homepage – www.bluej.org
17. MSDN: Object Test Bench – [msdn2. microsoft. com/ en-us/ library/ c3775d98 \(VS.80\). aspx](http://msdn2.microsoft.com/en-us/library/c3775d98(VS.80).aspx)
18. Cincom Smalltalk – www.cincom.com/visualworks