BISLs at the Research Group Software Development Methodology

Jan Dockx

Katholieke Universiteit Leuven, Department of Computer Science
Celestijnenlaan 200A, 3001 Leuven, Belgium
Jan.Dockx@cs.kuleuven.ac.be; http://www.cs.kuleuven.ac.be/

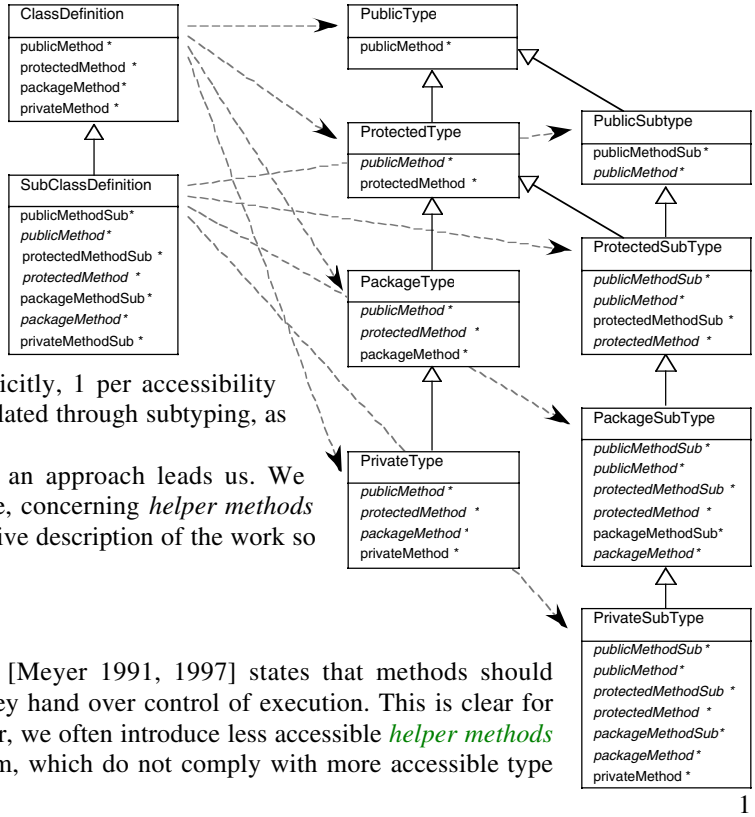
Presentation

Our research group is active in the domain of BISLs through education and research. We teach a second programming course [Dockx, Steegmans a, b] since 1997 in several academic programs and for the industry, using the *contract paradigm* [Meyer 1991, 1997] and *behavioral subtyping* [Liskov, Wing 1994] as the guiding paradigms, with formal specification in an Eiffel-like BISL. To support further research, we are developing an open source Java meta-model in the jnome project [Dockx, Mertens, Smeets, et al. a, b] we intend to use for BISL documentation and static verification. In [Dockx, van Dooren, Steegmans] we showed that internal iterators are the last step in the shift to object-oriented programming from procedural, structured programming, and dearly missed in Java. We offer a form of internal iterators for the Java Collection API in the jutil.org project [Blakeley, Dockx, et al.], which requires submissions to be specified formally using a BISL.

One of the topics we are researching currently is the definition of a clear semantics of non-public members. Below we report some initial findings of this research.

Accessibility & Helper Types

Users with more access, such as protected users, users in the same package, or users in the same compilation unit (private users), have access to more members. From experience we know that it is often desirable to extend public specifications of public members for users that have more access, so that they incorporate the effect on less accessible properties. Even more pressing is the need for the users with more access to know the preconditions that apply to more restricted properties. In other words, we would like to add a protected, package accessible, and private specification to public methods, next to the existing public specification; and make similar extensions to protected and package accessible methods.



The less accessible postconditions need to be extensions of the more accessible postconditions, and less accessible preconditions cannot contradict the more accessible preconditions. This sounds very much like *behavioral subtyping* [Liskov, Wing 1994]. It suggests that we should look at the more restricted parts of a class as *subtypes* of the less restricted parts. A Java class would introduce not 1, but 4 types implicitly, 1 per accessibility level. The *accessibility types* are related through subtyping, as shown in the figure.

We are examining where such an approach leads us. We want to report 2 partial results here, concerning *helper methods* and *protected users*. A more extensive description of the work so far can be found in [Dockx].

Helper Methods

In general the contract paradigm [Meyer 1991, 1997] states that methods should respect all type invariants when they hand over control of execution. This is clear for public methods. In practice however, we often introduce less accessible *helper methods* to encapsulate parts of an algorithm, which do not comply with more accessible type

invariants. Should this be allowed or not? Working with accessibility and implementation types strongly suggests that all methods should respect all possible type invariants. Type invariants can be introduced in `PublicType`, `ProtectedType`, `PackageType` and `PrivateType`. If these types are required to be behavioral subtyping compliant, type invariants in more accessible subtypes can only strengthen type invariants introduced in less accessible types. As a solution we propose *not to consider such helper methods to be part of the type introduced by the class*. We propose to add *helper types* next to the accessibility types. Helper methods can be defined there, together with the data structures they work on. A helper type instance can be used via delegation.

Subclasses

The figure shows how the accessibility types would behave when a subclass is defined outside the package. The package and private accessibility types are not related through inheritance. Method implementations in the subclass cannot see package accessible or private type invariants. They won't depend on them, but they also cannot be required to uphold them. In other words, the package and private implementation of the super class *needs to take care of them once and for all*.

Protected specifications of the superclass are presented to implementers of the subclass in 2 roles. They can call the public and protected methods defined in the superclass in their own implementations, depending on the contract specified for these methods in `PublicType` and `ProtectedType` and they can also strengthen their specification in `PublicSubType` and `ProtectedSubType`. Although the basis of the contract paradigm is that these 2 roles are separated, here they come together in 1 entity. This gives rise to interesting conflicts, which have surfaced often in our work in object-oriented frameworks.

When the specification in `PublicType` is not deterministic, often the superclass offers a *default implementation* that can be used by a subclass, or overwritten, as desired. The implementer of the subclass needs to know what the default implementation does to be able to make this choice. This postcondition can strengthen the original, non-deterministic postcondition of `PublicType` in `ProtectedType`. But, because this protected specification also is inherited into `ProtectedSubType`, the implementation in the subclass also needs to uphold it, making the default effect the only one allowed. This result is not what we hoped for.

Another kind of default implementation is a public method in an abstract superclass, which has an implementation that only partially reaches its contract. The intention is for subclasses to call the `super`-method, and do whatever else is needed to complete the contract before or after that call. When we use the accessibility types approach, this practice is forbidden, and rightfully so.

References

[Blakeley, Dockx, et al.] P. Blakeley, J. Dockx, M. van Dooren, E Steegmans: **jutil.org**; SourceForge; Labrador, Leuven; 2001; http://org-jutil.sourceforge.net/, http://www.sourceforge.net/projects/org-jutil/

[Dockx] J. Dockx:**Accessibility & Helper Types**; K.U.Leuven, Dept. of Computer Science; Leuven; 2002; CW 341; http://www.cs.kuleuven.ac.be/publicaties/rapporten/

[Dockx, Mertens, Smeets, et al. a] J. Dockx, K. Mertens, N. Smeets, M. van Dooren, E. Steegmans:**jnome**; http://www.jnome.org/

[Dockx, Mertens, Smeets, et al. b] J. Dockx, K. Mertens, N. Smeets, .E Steegmans:**A Java Meta Model in Detail**; K.U.Leuven, Dept. of Computer Science; Leuven; 2001; CW 323; http://www.cs.kuleuven.ac.be/publicaties/rapporten/

[Dockx, van Dooren, Steegmans] J. Dockx, M. van Dooren & E. Steegmans: **Dijkstra's Dream; Internal Iterators as Software Theorems**; Katholieke Universiteit Leuven, Dept. of Computer Science; Leuven; 2001; CW340; http://www.cs.kuleuven.ac.be/publicaties/rapporten/

[Dockx, Steegmans a] E. Steegmans & J. Dockx: **Objectgericht programmeren met Java**; Acco; Leuven; 2002?; ISBN 90-334-4535-2; in publication

[Dockx, Steegmans b] J. Dockx & E. Steegmans: **A New Pedagogy for Programming**; Sixth Workshop on Pedagogies and Tools for Learning Object Oriented Concepts, ECOOP 2002, June 11, 2002, Malaga, Spain; also avaible as CW Report: K.U.Leuven, Dept. of Computer Science; Leuven; 2002; CW 339; http://www.cs.kuleuven.ac.be/publicaties/rapporten/

[Liskov, Wing 1994] B. H. Liskov & J. M. Wing: **A Behavioral Notion of Subtyping**; *ACM Transactions on Programming Languages and Systems*, Vol. 16, Nr. 6; November 1994; p. 1811-1841

[Meyer 1991] B. Meyer: **Design by Contract**; *Advances in Object-Oriented Software Engineering*, Ed. D. Mandrioli, B. Meyer; Prentice Hall; Englewood Cliffs, N.J.; 1991; p. 1-50

[Meyer 1997] B. Meyer: **Object Oriented Software Construction**; 2nd Edition; Prentice Hall; Upper Saddle River, NJ; 1997; 1254 pages; ISBN 0-13-629155-4