Reuse Contracts: Connecting Bottom-Up and Top-Down Reuse

Position paper[1] submitted to OOPSLA 98 workshop on Object Technology, Architecture, and Domain Analysis — Experiences in Making the Connection

Kim Mens, Carine Lucas, Patrick Steyaert and Wilfried Verachtert
{kimmens, clucas, **prsteyae}@vub.ac.be**, **wilfried@MediaGeniX.com**
Programming Technology Lab
Vrije Universiteit Brussel, Belgium

Abstract

Whereas most object-oriented technologies traditionally achieve reuse in a bottom-up fashion, the research areas of architectures and domain analysis advocate a top-down approach to achieve *systematic* reuse. Practice shows that a combination of both is often desired or needed. The reuse contracts model can provide such a combination and can make the connection between object technology, architectures and domain analysis.

1. Position Statement

Traditional object technologies, with their emphasis on iterative development, allow the construction of reusable assets in a bottom-up fashion. Many reuse mechanisms and techniques are available, but reuse is not planned in advance; software systems are not designed *for* reuse. Reuse in OO is mostly ad hoc and lessons can be learned from the work on systematic reuse.

Domain analysis achieves systematic reuse by following a top-down approach in which the possible points of evolution (e.g. features, commonalities and variation points) are identified at the earliest phases of the life cycle and incorporated in the architecture and design. In this way, systems are built *for* reuse and are more robust to evolutionary changes. But unfortunately, most current domain engineering approaches do not take unanticipated or unforeseen reuses and changes into account.

We claim that the bottom-up approach (development with *reuse) of most object technologies needs to be reconciled with the top-down approach (development* for *reuse) of systematic reuse to make systematic reuse a standard practice. Such a reuse methodology must emphasise the co-operation between asset providers and asset reusers to control how and which assets* can be *reused, how assets* are *reused, and how (anticipated as well as unanticipated) changes propagate from provider to reuser assets during iterative development. We propose reuse contracts as the basis for such a methodology.*

2. Evolution is an inherent property of systematic reuse

Generally, the knowledge about the problem domain is realised through a set of components that are reusable within a formerly defined framework. A practical problem in defining the problem domain is that the domain knowledge usually isn t immediately available. An essential difficulty herewith is the impossibility to predict all possible variations and evolutions of the software (and the problem domain) beforehand.

Systematic reuse should not only recognise the need for a reusable asset to evolve both during its initial design and when it is being reused, it should actually advocate the development of a methodology for managing change in the process of engineering reusable software. The development of reusable assets is inherently an *evolutionary process*. A reuser can only gain insights in the qualities of reusable assets by

[1] This position paper is partially based on a similar position paper [Steyaert&al97] by the same authors presented at WISR 8, the 8th Annual Workshop on Software Reuse.

actually reusing them. A provider can only improve the qualities of assets if the experience of reuse is fed back to him. Successful assets can have a long life span and thus need to evolve and adapt to new reusers and their requirements. The inability to do so turns a reusable asset into legacy. Such iterative development is also important because it allows the construction of reusable assets in a bottom-up fashion. This is crucial for reuse to become economically feasible: it allows finding a delicate balance between the longer term investments needed for constructing reusable assets and the need to meet shorter term (customer) deadlines.

To be able to leverage on the investment made in building an asset, reusers must be able to benefit from future improvements of the assets they reuse: proper evolution of reused assets should not invalidate previous reuse. In a similar vein, reuse should go beyond the act of copying out code fragments and adapting them to current requirements without regard for the evolution of the reused fragments. This implies the management of some kind of consistency in the evolution of reusable software, to prohibit different versions of a reusable asset from propagating through different applications. While systematic reuse should present an opportunity to reduce maintenance effort, a proliferation of versions actually increases it, as older versions of an asset behave differently than newer versions. The absence of change management mechanisms is recognised as an important inhibitor to successful reuse [Goldberg&Rubin95], [Pancake95], [Yourdon94].

To reconcile the bottom-up approach of iterative development *with* reuse with the top-down approach of systematic reuse (development *for* reuse), we suggest a methodology that combines the best of both worlds by introducing systematic reuse in the object-oriented software engineering process. Our conjecture is that incrementally building reusable assets requires a strong co-operation between providers of reusable assets and asset reusers.

3. Reuse Contracts

In [Steyaert&al96, Lucas97, Mens&al98] reuse contracts are introduced as an object technology to manage (anticipated as well as unanticipated) reuses of assets at implementation, design or analysis level. Based on these results as well as on practical experiences with reuse contracts [Codenie&al97], we are confident that the reuse contracts approach is general enough to support and manage reuse of architectural building blocks as well.

The idea behind *reuse contracts* is that assets are reused on the basis of an explicit contract between the *provider* of an asset and a *reuser* that modifies this asset. The purpose of this contract is to make reuse more disciplined. For this purpose, both the provider and the reuser have obligations that are described in their contract clauses . The primary obligation of the provider is to document how the asset *can be* reused. The reuser needs to document how the asset *is* reused or how the asset evolves. Both the provider s and reuser s contract clauses must be in a form that allows to detect what the impact of changes is, and what actions the reuser must undertake to upgrade if a certain asset has evolved. Reuse contracts help in keeping the model of the provider consistent with the model of the reuser.

Before the provider can document evolution, he needs to document what properties of the asset can be relied on at a particular point in time. The *provider clause* states certain properties of the entities in the provided asset. In the *reuser clause*, the reuser documents the changes made to the provided asset. This is achieved by linking a reuse contract s provider and reuser with a *contract type*. The contract type expresses *how* the provided asset is reused. The contract type imposes obligations, permissions and prohibitions onto the reuser. For example, the extension contract type obliges reusers to add new elements, but prohibits overriding of existing elements. It permits adding multiple elements at once. Contract types and the obligations, permissions and prohibitions they impose are fundamental to disciplined reuse, as they are the basis for detecting conflicts when provided assets are reused.

In a reuse contract, the provider clause provides only a certain view on the actual asset. Thus an asset can participate in different reuse contracts that address different concerns of the provided asset. Typical examples of general concerns are persistence, distribution and user interaction.

Originally, reuse contracts were used at the implementation level to express reuse in evolvable class inheritance hierarchies [Steyaert&al96], and reuse and evolution of collaborating classes [Lucas97]. More recently, work has been done on integrating the reuse contract formalism into the Unified Modelling

Language [Mens&al98]. In order to actively apply the reuse contract methodology, however, a lot of work still needs to be done. Currently we focus on the integration of the different incarnations of the reuse contract model, on the application of reuse contracts on a higher architectural level and the integration of reuse contracts in the development process [De Hondt98].

4. Related Work

In this section we compare some of the strengths of reuse contracts with other object-oriented technologies.

4.1 Frameworks, Design Patterns and Cookbooks

Most state-of-the-art object-oriented reuse methodologies are insufficiently formally supported. This is e.g. the case for object-oriented frameworks that are usually documented through very informal documents. One way to additionally document frameworks is through design patterns, but these suffer the same lack of formal underpinnings. Cookbooks that guide reusers step by step in the reuse process (the customisation process for frameworks) were suggested as another approach, but these suffer from the additional problem that they are overly coercive, i.e. they can only guide reuses that were specified up front. Reuse contracts provide a formally underpinned documentation that still allows enough flexibility.

4.2 Facades and Variation Points

While, up until now, the issue of systematic reuse was almost entirely neglected in object-oriented analysis and design techniques, recently new concepts such as facades and variation points [Jacobson&al97] were introduced to tackle this problem. These approaches try to discover possible points of reuse at the earliest phases of the life cycle and incorporate them in the design. While a good starting point, we believe that a complete methodology should also incorporate the handling of unanticipated reuses. That concern is one of the main contributions of reuse contracts to these other methodologies.

4.3 The Unified Modelling Language

UML provides little support for modelling evolvable or reusable specifications and designs. It is explained in [Mens&al98] how UML can be enhanced with support for reuse and evolution of model components consisting of collaborating classes based on the reuse contract formalism. Among others, this gives us a formal semantics for reuse of UML model components that allows us to detect evolution and composition conflicts semi-automatically.

5. Conclusion

In this paper we argued how domain analysis methods and object-oriented technologies can meet each other half way in reconciling top-down and bottom-up reuse. Reuse contracts were presented as a candidate to help bridge this gap. While a promising approach, a number of questions still need to be addressed. Interesting topics for further research include the integration of the different incarnations of the reuse contract model, the application of reuse contracts on a higher architectural level and the integration of reuse contracts in the development process.

6. References

[Codenie&al97] W. Codenie, K. De Hondt, P. Steyaert and A. Vercammen. *From Custom Applications to Domain-Specific Frameworks*. Communications of the ACM, Special Issue on Application Frameworks, 40(10), pp. 70-77, October, 1997.

[De Hondt98] Koen De Hondt, *A Novel Approach to Architectural Recovery in Evolving Object-Oriented Systems*, PhD Dissertation, Vrije Universiteit Brussel, 1998.

[Goldberg&Rubin95] A. Goldberg and K. Rubin, Succeeding with Objects: Decision Frameworks for Project Management, Addisson-Wesley, 1995.

[Jacobson&al97] I. Jacobson, M. L. Griss, and P. Jonsson. Software Reuse: Architecture, Process and Organization for Business Success, Addisson-Wesley, 1997.

[Lucas97] C. Lucas. Documenting Reuse and Evolution with Reuse Contracts. PhD Dissertation, Vrije Universiteit Brussel, September 1997.

[Mens&al98] T. Mens, C. Lucas and P. Steyaert. Supporting Reuse and Evolution of UML Models. Position paper at the UML '98 Workshop, Mulhouse, France, 4 June 1998.

[Pancake95] C. Pancake. Object Roundtable, The Promise and the Cost of Object Technology: A Five-Year Forecast, Communications of the ACM, 38(10):32--49, October 1995.

[Steyaert&al96] P. Steyaert, C. Lucas, K. Mens and T. D'Hondt. *Reuse Contracts: Managing the Evolution of Reusable Assets*. Proceedings of OOPSLA '96, ACM SIGPLAN Notices, 31(10), pp. 268-286, ACM Press, 1996.

[Steyaert&al97] P. Steyaert, C. Lucas and K. Mens. Reuse Contracts: Making Systematic Reuse a Standard Practice. Proceedings of WISR 8, the 8th Annual Workshop on Software Reuse, at Ohio State University, March 23-26, 1997.

[Yourdon94] E. Yourdon, Object-Oriented System Design: An Integrated Approach, Yourdon Press Computing Systems, Prentice Hall, 1994.