Applying Reuse Contracts in a Product Line Approach

Presented at the OOPSLA'98 Workshop on Object Technology and Product Lines

Carine Lucas[+], Kim Mens[+], Patrick Steyaert[+], Wilfried Verachtert[*]

Email: clucas@vub.ac.be, kimmens@vub.ac.be, prsteyae@vub.ac.be, wilfried@MediaGeniX.com

[+]Programming Technology Lab
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
http://progwww.vub.ac.be/

[*]MediaGeniX
Otto de Mentockplein 19
1853 Strombeek-Bever, Belgium

Introduction

Our research on reuse contracts demonstrated how a better documentation of the dependencies between reusable assets and their reusers can help in managing the evolution of reusable assets and in achieving disciplined reuse. However, in trying to apply our results to product lines we still encounter multiple problems. This position paper shortly introduces reuse contracts and then raises some topics of research we feel are crucial in order to set up successful product lines.

Reuse Contracts

The essential idea behind reuse contracts is that a component is reused on the basis of an explicit contract between the *provider* of the component and a reuser that *modifies* this component. The purpose of a contract is to make reuse and evolution more disciplined. For this purpose, both the provider and the reuser have *contractual obligations*. The primary obligation of the provider is to document how the component can be reused. The reuser needs to document how the component is reused or how the component evolves. Both the provider's and the reuser's documentation 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 component has evolved. To summarise we can say that reuse contracts help in keeping the model of the provider consistent with the model of the reuser.

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 in product lines 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]. A number of the questions raised are discussed below.

How to support component evolution ?

Experience shows that the development of useful components is an evolutionary process. Only rarely a newly developed component can be immediately used in another context. Only through consequent reuses of a component and adoptions to different cases a component becomes mature. The support of *incremental and iterative development* is therefore crucial. Today we are able to realise this incremental and iterative development on the different levels of the software life cycle. However, the translation of an increment on one level to another, for example from analysis to design to implementation, remains a difficult issue. The lack of a formal description of this translation causes the different layers to continuously drift away from each other. As a result, the only evolving – and thus useful – components are those on the implementation level. *Traceability* between the different layers of the software life cycle is therefore an absolute minimum.

One of our current research topics is therefore to investigate how the reuse contract types that are used to describe dependencies between components and their reusers can be used to describe the relationship between components in different phases of the life cycle.

How to build software by composing reusable components ?

Product lines aim to build software by composing different components. The question how these existing software components interact is still not adequately addressed. One of the reasons for the remaining problems is that de developers of software components make a lot of *implicit assumptions* about how a component needs to co-operate with other components. Often these assumptions can only be discovered after inspection of the code, which is an error-prone and time-consuming process. Therefore, it is crucial to make as much of these assumptions as possible explicit in as early a stage as possible. A second problem concerns the so-called *architectural mismatch* [Garlan&al95]. When different software components are constructed in different architectural styles, the problem is how these different styles can be integrated.
We are therefore investigating what kind of information on developer's intentions and architectural styles could be useful to integrate in the reuse contract method, in order to be able to detect more composition and evolution conflicts.

How to document the architecture of an evolving system ?

A prerequisite for effective reuse is a thorough understanding of the system that is reused. In particular, a good understanding of the architecture - the organization of the source code as composition of components and the interaction between them - is crucial. Although softwaredocumentation should help with acquiring a better software comprehension, it seldom does, especially for evolving systems. Software documentation is never up-to-date, not in an appropriate form for the development task at hand, or in many cases even non-existent. Software documentation is hard to keep up-to-date because it is not integrated in the software development environment and hence not resistant to change. In our current research we are investigating novel approaches to architectural recovery of object-oriented systems. The techniques explored in [De Hondt98] are based on software classification as a uniform organizational structure to record architectural descriptions as tangible software entities in the development environment.

Conclusion

While a lot of new developments seem promising towards the construction of product lines, a lot of technical issues still need to be addressed. This paper raised some questions which we are interested to discuss during the workshop.

References

[Garlan&al95] David Garlan, Robert Allen and John Ockerbloom, *Architectural mismatch: why reuse is so hard*", IEEE Software, 12(6), pp. 17-26, November 1995.

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

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

[Mens&al98] Tom Mens, Carine Lucas and Patrick Steyaert, *Supporting Reuse and Evolution of UML Models*, Proceedings of UML'98 International Workshop, Mulhouse, France, June 1998

[Steyaert&al96] Patrick Steyaert, Carine Lucas, Kim Mens, Theo 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.