Spectrum: A Formal Approach to Software Development with Reusability

Martin Wirsing
Bayer. Forschungszentrum für Wissensbasierte Systeme
Universität Passau
wirsing@forwiss.uni-passau.de

This research has been partially sponsored by the DFG-project Spectrum, the BMFT-project Korso and the ESPRIT working group COMPASS.

Abstract

A formal approach to software development is presented which integrates reuse as an essential part: the Spectrum approach. Based on type-theoretic, algebraic and functional methods, the Spectrum project aims at studying methodical aspects, language issues, theory and support tools of software reuse in an integrated way. A central concept is the notion of software component which represents software at different levels of abstraction and gives access to multiple views of software objects. The type-theoretic meta-language serves for specifying exactly the properties of such components, algebraic methods are used for carrying out formal developments of specifications.

Keywords: algebraic specification, reusable software components, software development, type-theory

1 Introduction

The increasing demand for more and more complex software systems makes it necessary to change the process of software development. It is important to reuse already developed software systems for different applications. In order to achieve this goal and to get the confidence of the system developers, reuseable software must be of high quality.

Therefore formal methods become an integrated part of the software construction process: formal methods provide abstract language concepts for supporting the certification of software; language specific theory of reuse provides the foundations for computer aided development with reuse and formal methods support the evolution of reusable objects and other reuse activities such as construction, search and classification, integration and configuration of reusable components. Moreover

the integration of formal methods into the software life cycle is becoming an important research issue [Dort 91].

In the following a formal approach to software development will be presented which integrates reuse as an essential part: the Spectrum approach has its roots in the CIP project [Baue 81] and several ESPRIT projects such as METEOR (on algebraic specification and development), PROSPECTRA (on transformational techniques) and DRAGON (on reusability).Based on type-theoretic, algebraic and functional methods, the Spectrum project aims at studying methodical aspects, language issues, theory and support tools in an integrated way.

2 Methods

2.1 Software components

A central aspect of the Spectrum approach is the notion of software component which represents software at different levels of abstraction and gives access to multiple views of software objects. Starting from a simple model of the software development process where a development consists of a sequence of descriptions beginning with a requirement specification and ending with the final program, we define a software component to be a finite graph whose nodes are software objects such as specifications and programs and whose edges represent relations between such objects. Then each development sequence can be understood as a particular software component.

2.1.1 Objects

Software objects are specifications of functional and non-functional properties, programs and components themselves. In general, objects are structured. In particular, structured specifications can be considered as software components related by operations such as inheritance, clientship or instantiation.

Objects come together with associated attributes such as examples of instances, theorems, meta-theorems, simulations and different syntactical representations [Lato 91].

2.1.2 Relations between software objects

There are different kinds of relations:

a) Refinement and implementation relations relating objects at different levels of abstraction.

b) Translations between expressions in different languages. Translations are especially important for the transition from specifications to programs and for the reuse of results from other languages. For example, for a development in a first-order logic it might be interesting to use results obtained in equational logic with a narrowing-based theorem prover.

c) Multiple views of software objects describe functional properties (such as input-output behaviour, invariants and theorems) and non-functional properties (such as concurrency aspects, complexity mesures and access control) [Cazi 91].

Relations are classified according to their abstract properties. Most important are transitivity and monotonicity which insure the correctness of a sequence of development steps as well as of local development steps [Broy 80, Baxt 90].

2.2 Development for and with reuse

Software development is understood as the development of a (part of a) sofware component. Software components can be manipulated from three different views [Stab 91]:

a) The reuse supervisor maintains the reuse components and is the only one who has "write" access to the component library.

b) The reuse consultant knows about the content of the whole library and is able to share his knowledge with "ordinary" reusers.

c) Each reuser has a partial view of the library and constructs particular applications using his view. The approach also supports the development of software by a group of reusers.

For each view different operations are allowed:

a) The reuse supervisor has the right to release new views of the component. He may extend existing views and replace parts within a view.

b) A reuse consultant may inspect the whole library and tell the supervisor that a view needs restructuring.

c) Ordinary reusers may produce new versions of their view by extending, combining and changing views and may prepare a new version for submission to the supervisor. Cooperation between several reusers is achieved by formal support for the integration of software objects and by formal support for conversation between different developers [Rose 91].

3 Language

In Spectrum three different language styles are studied: Type theory, algebraic specification and functional programming.

3.1 Type theory

As a meta-language for formally expressing components and software developments an extension of ECC [Luo 91] is used. This is a very powerful calculus in which module concepts and polymorphism can be expressed adequately with the help of dependent products and dependent sums. A hierarchy of universes allows to describe programming and meta- programming as well as reasoning and meta-reasoning in the same framework.

3.2 Algebraic specification

For describing design specifications in an algebraic style the kernel language ASL [Sann 83] and their dialects RAP [Gese 86] and OS [Breu 91] are used. ASL provides a few simple but powerful specification operators for writing specifications in a structured way and posesses a well developed theory for transforming specification expressions and proving properties in a structured way. RAP allows to write hierarchical specifications and is embedded in a convenient development environment. OS integrates object oriented features into ASL and is well-suited for specifying object oriented programs.

Algebraic specifications do not support dependent types and therefore are less expressive than type theoretic descriptions. Advantages of algebraic specifications are that they are abstract, easy to manipulate and that they have a well-developed theory together with good programming environments.

3.3 Functional programming

SML [Miln 90] is used as functional programming language. SML supports modular and high-level functional programming and integrates non-functional programming styles (such as references, assignments and concurrency) in a clean and transparent way. SML structures correspond to algebras and can therefore be understood as models of equational specifications. Moreover its module concepts can be easily expressed in the type theoretic framework. SML is used for constructing prototype implementations of specifications.

4 Theory

The theoretical background of Spectrum can be divided into two main streams: Foundational studies and theory of reuse.

4.1 Foundational studies

The foundational studies concern semantical properties of specification and type-theoretic languages.

4.1.1 Algebraic specifications

For algebraic specifications a well-developed theory supporting formal developments is available [Wirs 90]. In particular, the fundamental implementation relation is transitive and monotonic and therefore satisfies the horizontal and vertical composition property. Using the equations of "module algebra" structured specifications can be transformed into various normal forms and structured proof calculi allow one to verify the correctness of implementations in a structured way [Wirs 91]. Context induction provides a new method for verifying behavioural implementations [Henn 90]. Most of these results can be extended to object oriented specifications [Breu 91].

4.1.2 Non-functional specifications

For non-functional specification two basic methods are available: In [Baxt 90] a performance measure is associated with the abstract syntax of program. Then any non-functional specification is given as the conjunction of preorder assertions between performance expressions. [Jacq 91] advocates the abstract interpretation of specification expressions in order to perform complexity analysis or measurements and benchmarks.

4.1.3 Integration of formalisms

For relating programs and specifications there are two basic semantic approaches. By considering "programs as specifications" one defines a translation from a subset of specification expressions to programs. For example, equational specifications with axioms in the form of structural recursive definitions can be directly transformed into functional programs. In the type-theoretic approach it seems appropriate to consider programs as "realizers" of specifications, i.e. the semantics of a program is considered as a model of the specification.

More generally, type-theoretic descriptions are well-suited for integrating different styles of specifications and programs as has been shown by the DEVA project [Cazi 91].

4.2 Theory for reuse

Based on the results of the foundational studies the theory of reuse provides formal support of most reuse activities. In the Spectrum project we are mainly interested in the follwing issues.

4.2.1 Reuse of components

The construction of reusable components is based on the theory of structured algebraic specifications. In [Wirs 88] a notion of reusable component is developed that represents specifications at different levels of abstraction by a tree of specifications where two consecutive nodes are related by the implementation relation. Due to the fact that the nodes are ASL specifications these components can be normalized and enjoy vertical and horizontal composition properties. We are currently studying how these results can be extended to software components consisting of graphs of specifications and how functional and non-functional properties can be expressed in a type-theoretic framework.

Signature matching and the classificaton of specifications by the implementation relation help in retrieving components from a library. After matching a given abstract problem specification SP with a specification in the library, an implementation of SP can be automatically constructed from the implementation found in the library. If the matching is correct then due to the composition properties the new implementation is correct without requiring any further proof.

In [Henn 91] a formal method has been developed for integrating several module implementations to a consistent configuration of the whole system.

4.2.2 Reuse of developments

For the reuse of development steps there are two main approaches. [Part 91] and [Baxt 90] use meta-descriptions of transformations written in a first order language whereas [Cazi 91] take the introduction and elemination rules of the lambda-typed lambda- calculus underlying DEVA. Baxter studies the difficult problem of pruning development histories in order to delete non-reusable steps. Then the sequence of development steps is repaired by integrating socalled "maintenance deltas". [Rose 91] uses the object oriented design language Telos for formally describing the design decisions occurring in a development.

5 Support tools

Up to now only tools for the development of specifications have been implemented in the Spectrum project.

The RAP&TIP system is an environment for prototyping hierarchical algebraic specifications; moreover it includes an inductive theorem prover for equational formulas [Frau 91].

The ISAR system gives support for the correctness proof of behavioural implementations [Baue 91]. The main technique is context induction which is performed with the help of TIP.

Components of RAP specifications can be designed interactively with the help of the Specifiers Notepad [Breu 90].

Acknowledgement

Thanks go to Stefan Gastinger, Michael Gengenbach, Rolf Hennicker, Robert Stabl (reusability group of Spectrum) and to Bernhard Reus, Thomas Streicher (semantic group of Spectrum) for inspiring discussions and many ideas which are reflected in this paper. Thanks go also to the working group on formal methods of the First Int. Workshop on Software Reuseability (Dortmund 1991) and in particular to Rene Jaquart and Larry Latour for many new insights into formal support for software reuse.

References

[Baue 91] B. Bauer: Ein interaktives System fuer beobachtungsorientierte Implementierungsbeweise. Diplomarbeit, Universitaet Passau 1991.

[Baue 81] F.L. Bauer, M. Broy, W. Dosch, R. Gnatz, B. Krieg-Brueckner, A. Laut, M Luckmann, T.A. Matzner, B. Moeller, H. Partsch, P. Pepper, K. Samelson, R. Steinbrueggen, M. Wirsing, H. Woessner: Programming in a wide spectrum language: a collection of examples. Science of Computer Programming 1, 1981, 73-114.

[Baxt 90] I.D. Baxter: Transformational maintenance by reuse of design histories, Ph.D. Thesis, University of California, Irvine, Tech. Report 90-36, 1990.

[Breu 91] R. Breu: Algebraic specification techniques in object oriented programming environments. Dissertation, Universitaet Passau, 1991, also to appear in Springer Lectures Notes of Computer Science.

[Breu 90] R. Breu, H. Windl: The specifiers notepad - a hypertext system tailored to the design of algebraic specification, Tech. Report, Universitaet Passau, MIP-9007, 1990.

[Broy 80] M. Broy, H. Partsch, P. Pepper, M. Wirsing: Semantic relations in programming languages. In: S. Lavington (ed.): Proc. IFIP World Congress - Information Processing 80, Amsterdam: North Holland 1980, 101-106.

[Cazi 91] J. Cazin, P. Cros, R. Jacquart, M. Lemoine, P. Michel: Construction and reuse of formal program developments. In: S. Abramski, T.S.E. Maibaum (ed.): TAPSOFT 91, Springer Lectures Notes in Computer Science 494, 1991, 120-136.

[Dort 91] W. Schaefer (ed.): Proc. First Int. Workshop on Software Reusability, Dortmund, 1991.

[Frau 91] U. Fraus, H. Hussmann: A narrowing-based theorem prover. In: Proc. RTA 91, Springer Lectures Notes in Computer Science 488, 1991.

[Gese 86] A. Geser, H. Hussmann: Experiences with the RAP system - a specification interpreter combining term rewriting and resolution. Proc. European Symposium on Programming. Springer Lecture Notes in Computer Science 213, 1986, 339-350.

[Henn 90] R. Hennicker: A proof principle for behavioural abstractions. In: A. Miola (ed.): Proc. DISCO 90. Springer Lecture Notes in Computer Science 429, 1990, 101-110.

[Henn 91] R. Hennicker: Consistent configuration of modular algebraic implementations. Tech. Report, Universitaet Passau, MIP-9102, 1991.

[Jacq 91] R. Jacquart: Reuse of formal developments. In: [Dortmund 91]

[Lato 91] L. Latour: In: [Dortmund 91] .

[Luo 91] Z. Luo: A unifying theory of dependent types. Tech. Report, University of Edinburgh, ECS-LFCS-91-154, 1991.

[Miln 90] R. Milner, M. Tofte, R. Harper: The definition of Standard ML. London: MIT Press, 1990, 101 p..

[Part 91] H. Partsch, N. Voelker: Another case study on reusability of transformational developments - Pattern matching according to Knuth,Morris and Pratt. In: M. Broy, M. Wirsing (eds.): Methods of Programming, Springer Lecture Notes in Computer Science, to appear.

[Rose 91] T. Rose: Entscheidungsorientierte Versionen- und Konfigurationenverwaltung. Dissertation, Universitaet Passau, 1991.

[Sann 83] D. Sannella, M. Wirsing: A kernel language for algebraic specification and implementation. In M. Karpinski (ed.): Colloquium on Foundations of Computation theory. Springer Lecture Notes in Computer Science 158, 1983, 413-427.

[Stab 91] R. Stabl: Personal communication, September 1991.

[Wirs 88] M. Wirsing: Algebraic description of reusable software components. In: Proc. COMPEURO '88, Computer Society Press of the IEEE, no. 834, 300-312, 1988.

[Wirs 90] M. Wirsing: Algebraic specification. In: J.van Leeuwen (ed.): Handbook of Theoretical Computer Science, Vol. B, Amsterdam: North-Holland, 1990, 675-788.

[Wirs 91] M. Wirsing: Stuctured specification: syntax, semantics and proof calculus. In: H. Schwichtenberg (ed.): Proc. International Summer School Marktoberdorf 1991, to appear.

6 About the Author

Prof. Dr. Martin Wirsing studied Mathematics at the University of Paris and at the Technical University of Munich. From 1975 until 1983 he worked as a research and teaching assistant at the Sonderforschungsbereich "Programmiertechnik" at the Technical University of Munich. During this time he had been a member of the Munich CIP group that worked on program specification and transformation. He wrote his dissertation at the Technical University of Munich (with the title "Das Entscheidungsproblem der "adikatenlogik erster Stufe mit Funktionszeichen in Herbrandformeln") and his habilitation (1984 with the title "Structured algebraic specifications: a kernel language"). Since 1985 he is full professor for Computer Science at the Faculty of Mathematics and Computer Science at the University of Passau. He is working in the ESPRIT projects METEOR, DRAGON and in the ESPRIT Working Group COMPASS. From 1986 to 1988 he had been the chairman (Dekan) of the Faculty of Mathematics and Computer Science of the University of Passau, and from 1988 till 1990 he was the vicechairman (Prodekan) of the faculty. Martin Wirsing has published more than 90 scientific papers in the areas of mathematical logic, programming methodology, semantics of programming languages, program transformation and program development, formal specification and algebraic specification languages. Since 1990 he is one of the directors of the "Bavarian Research Center of Knowledge-based Systems" (FORWISS). He is member of the editorial board of several scientific journals including Theoretical Computer Science, Technique et Science Informatique, RAIRO-Informatique Theorique et Applications and Research Notes in Theoretical Computer Science.