Final Scientific Report of
A Framework Approach to Composing Heteregeneous Applications
NFS Project No. 20-53711.98

15 November 2000

1 Results Description

The results presented below are grouped into four closely related research areas:

1. the PICCOLA language, which investigates heteregeneous component composition,
2. the COLAS language, which investigates coordination of active and distributed objects,
3. the building of frameworks and
4. the reengineering of object-oriented applications.

The four areas are complementary. The research led around the PICCOLA and COLAS languages focuses on models, mechanisms and techniques for composing and coordinating components in heterogeneous and distributed environments. Both theoretical and practical results were achieved in these two foundational tracks. As components are not composable per se but require an explicit composition architecture, our research into frameworks has focused on the complex issues involved in defining suitable architectural support. While the first three research efforts are basically top-down, the fourth track is bottom-up, and tries to answer the question, *How can we transform industrial object-oriented legacy code into more flexible, component-based applications?* Here we have developed a series of tools to support reengineering of object-oriented systems towards components, and documented a catalogue of best-practice patterns for reverse- and reengineering

2 PICCOLA

PICCOLA is a small *composition language* that embodies the paradigm "Applications = Components + Scripts" [AN00a]. PICCOLA models components and composition abstractions by means of a unifying foundation of communicating concurrent agents. Flexibility and extensibility are obtained by modelling both interfaces to components and the contexts in which they live by extensible records, or *forms*. PICCOLA supports software composition by means of *compositional styles* which express the kind of components and connectors that are relevant to a particular application domain.

We have obtained the following results:

- $\pi\mathcal{L}$ *Foundations.* PICCOLA was originally based on the π-calculus since this was the most promising foundation for modeling dynamically evolving concurrent systems of components. Our attempts to model composition abstractions (like mixins and inheritance) with the π-calculus led us to conclude that *forms* (extensible records) are a better basis for modeling composition than tuples, and this led to the development of the $\pi\mathcal{L}$-calculus [Lum99, LAN00, Sch99, SL00].

- *Language design.* It turns out that forms can be unified with abstractions. This allows us to simplify and reduce the language. We iterated on the syntax definition and made also use of the fact that forms can represent both the static and dynamic environment in a programming language [ALSN00, AN00a]. It should be noted that the current version of PICCOLA has only three keywords (def, root, and dynamic) and 8 built-in operators. We interpret this as validation for the compactness and simplicity of the PICCOLA language design. Abstractions for flow-control like if-then-else or loop constructs can directly be defined within PICCOLA. Using the dynamic and static environment, it is also possible to define abstractions for exception handling [AN00b]. This fact allows us one side to keep the language small and it also serves as an indication that higher order composition abstractions can in fact be implemented.

- *Compositional styles.* PICCOLA encourages the separation of stable and flexible parts of an application into components and scripts. We use this separation to support software evolution. The separation is made explicit by the use of compositional styles. A compositional style [Sch99, SN99] formalises composition by defining component types, connectors, and rules governing composition. We have implemented styles for scripting pipes and filters, actors, regulated actors [AKN00], events and gui-composition [AN00a], and

mixin-layer composition.

- *Aspect-oriented programming.* PICCOLA can directly express *aspects* as abstractions [Ach00, NA00]. This also gives evidence that forms and form abstraction are the right core mechanisms to give a language the expressive power needed to define arbitrary composition mechanisms.

- *Implementations.* We replaced tuples by forms in PICT and implemented a first prototype for PICCOLA [Ach98]. The language has evolved considerably since the first prototype, and PICCOLA has now been implemented in Java, Squeak, and Delphi. The same PICCOLA infrastructure is used to compose components conforming to different component models. Piccola prototypes are available at http://www.iam.unibe.ch/∼scg/Research/

We are continuing to develop PICCOLA and conduct practical experiments with it. We have started work on distributed PICCOLA as part of the continuation of this project. A long-term goal is to be able to support reasoning about compositions of components based on properties of components and compositional styles. We believe that PICCOLA and the notion of compositional style can be fruitful applied to specify, implement, and reason about distributed and mobile agents of todays Internet application [NSA00]. To this end we are now working on a higher-level, direct semantics of PICCOLA that will better support reasoning than the relatively low-level foundation of the $\pi\mathcal{L}$-calculus.

3 CoLaS: Coordinating Active Objects

In current object-oriented languages the code responsible for coordinating active and distributed objects is invariably tangled with the computational behaviour. As a consequence, objects suffer from a lack of abstraction of coordination aspects, and their implementation is polluted with external concerns like coordination and distribution.

Hi Stef: Please check. I rephrased for readability, but may have introduced some inaccuracies. – oscar

- *Language Design.* We have specified and implemented a language named COLAS that allows us to express the coordination of active and distributed objects as first class entities [CD99b, CD99a]. COLAS is based on a minimal set of abstractions necessary to support coordination. Instead of developing an entirely new language, we decided to extend an existing object-oriented language, Smalltalk, by introducing suitable coordination abstractions.

In COLAS the coordination is based on the notion of *Coordination Groups* that encapsulate all the information necessary for coordinating active objects. A group specifies the participants of the coordination in terms of participant roles, the coordination state, and the protocol for coordinating the participants.

Since message passing is at the core of object-oriented programming and coordination is essentially a matter of controlling object interactions, we evaluated how message passing control could be implemented effectively in Smalltalk, the host language of COLAS [Duc99].

- *Implementation.* A prototype of the COLAS language is currently implemented in VisualWorks and Distributed Smalltalk using Corba 2.0 services and the I3 protocol offered by Distributed Smalltalk. COLAS is available at http://www.iam.unibe.ch/∼scg/Research/

4 Coordination Frameworks

Components are not composable just because they are components. They have to be used in a context where they can be composed. We name such a context a composition architecture or composition framework. To gain understanding and experience we built several frameworks in the areas of coordination of components.

- *Design Guidelines.* In [TCD00] we present a series of guidelines for developing component frameworks in which coordination is an issue.

- *Coordination Medium.* In [Küh98] we present a coordination medium for distributed application that extends the notion of *tuple spaces* to *form spaces*, with corresponding advantages for flexibility and extensibility.

- *Customizable Coordination Frameworks.* In [DHN00], we present, OPEN SPACES, a framework for building families of tuple spaces based applications. OPEN SPACES defines a minimal but extensible core for building tuple-space based applications. Hotspots can then be specialized to introduce new functionality.

- *Implementations.* OPEN SPACES is implemented using VisualWorks and Distributed Smalltalk and is available at http://www.iam.unibe.ch/∼scg/Research/

5 Reengineering of Object-Oriented Applications

In parallel to our work on composing applications from heterogeneous and distributed components, we are conducting research into transforming object-oriented legacy systems into more flexible applications based on components. Whereas the first three tracks can be considered as being "top-down", this last track is essentially bottom-up, focusing on providing solutions that fit industrial constraints and situations.

Our contributions in the domain of reengineering object-oriented systems include:

Hi Stef: I removed the names of individual people, and rephrased some text. Please check. – oscar

- *Definition of a language independent meta model (FAMIX).* We needed to analyse several different object-oriented languages such as Smalltalk, Java, C++. We designed an extensible and language-independent meta-model for representing the main elements of object-oriented programming languages. [DDT99b]

- *Implementation of a reengineering environment (MOOSE).* To support our research we developed a reengineering environment based on the meta-model we specified. It includes the possibility to analyse several models, to define dedicated program analysers, to compute metrics, to load and save meta-models from different languages. [TCD00, DLT00, TDD00, TD99, Dem99]

- *Evaluation of metric use in reengineering.* We evaluated how software metrics can support a reengineering effort. From our studies, we concluded that metrics are not reliable to detect design flaws, that metrics are a good indicator of systems stabilisation, and that they can be used to recover refactorings. [DD99, DDN00a, DLS00]

- *Reverse Engineering in the Large.* We developed an approach to support the reverse engineering of large systems. The idea is to display software entities as nodes of trivial graphs but to semantically enrich the obtain graphs with metric values of the represented entities. [DDL99]

  The approach has been validated by the implementation of CodeCrawler, a metrics visualization tool that is based on MOOSE, and allows one to reverse engineer large systems [Lan99].

- *Detection of code duplication.* We developed an approach to identify duplicated code. Our approach is lexical thus limiting the language dependence. [DRD99, DRG99]

This approach has been validated by the development of DUPLOC, an interactive tool for detecting and visualizing patterns of duplicated code in large software systems.

- *Use of Dynamic information for extracting views and roles.* We developed an iterative approach for extracting architectural views of software systems based on the analysis of both static and dynamic information represented using the meta-model we developed. Views of a system can be created and refined incrementally. Moreover, we used dynamic information to support the extraction of design roles. [RD99, Ric99]

  This approach has been validated by the implementation of GAUDI, a tool to support program understanding based on Famix and MOOSE.

- *Reengineering Patterns.* Reengineering projects, despite their diversity, typically encounter the same problems and solutions again and again. We have defined a pattern form to transfer reengineering expertise and record reengineering patterns. Reengineering patterns codify and record knowledge about modifying legacy software: they help in diagnosing problems and identifying weaknesses which hinder further development of the system and aid in finding solutions which are more appropriate to the new requirements. Reengineering patterns are stable units of expertise which can be consulted in any reengineering effort: they describe a process without proposing a complete methodology. [DDT99a, DDN00b, DDN00c, DRN99]

- *Experiences with Design Extraction.* We started some experiments with the extraction of design information from the code. (Project connect Rose and MOOSE, Project DoMe-Moose)

6 Publications

The publications listed below are for the period from the beginning of the project until September 30 2000. The publications in **bold** are included with this report and cover the period from September 30 1998 to September 30 2000.

From October 1996 to October 1999, our team participated in the FAMOOS Esprit Project 21975 (Framework based Approach for Mastering Object Oriented Systems, BBW Nr 96.0015). While the project ended in October 1999, we continued to work on the reengineering of object-oriented legacy systems. The list of NFS publications below includes only new results which have been published *since* the end of the FAMOOS project.

We separately include a list of relevant papers published *before* October 1999, and which have already been included in the final FAMOOS report to the BBW. These papers are listed here for information only, and should not be considered as deliverables for this NFS project.

NFS Related Publications

[Ach98] Franz Achermann. Jpict - a framework for pi agents. tech. note, IAM, U. Berne, November 1998.

[Ach00] Franz Achermann. Language support for feature mixing. In *Workshop on Multi-Dimensional Separation of Concerns in Software Engineering (ICSE 2000)*, Limerick, Ireland, June 2000.

[AKN00] Franz Achermann, Stefan Kneubuehl, and Oscar Nierstrasz. Scripting coordination styles. In António Porto and Gruia-Catalin Roman, editors, *Coordination Languages and Models*, LNCS 1906, pages 19–35, Limassol, Cyprus, September 2000.

[ALSN00] Franz Achermann, Markus Lumpe, Jean-Guy Schneider, and Oscar Nierstrasz. Piccola – a small composition language. In Howard Bowman and John Derrick., editors, *Formal Methods for Distributed Processing, an Object Oriented Approach*. Cambridge University Press., 2000. to appear.

[AN00a] Franz Achermann and Oscar Nierstrasz. Applications = Components + Scripts – A tour of Piccola. In Mehmet Aksit, editor, *Software Architectures and Component Technology*. Kluwer, 2000. to appear.

[AN00b] Franz Achermann and Oscar Nierstrasz. Explicit Namespaces. In Jürg Gutknecht and Wolfgang Weck, editors, *Modular Programming Languages*, LNCS 1897, pages 77–89, Zurich, Switzerland, September 2000.

[CD99a] Juan-Carlos Cruz and Stéphane Ducasse. Coordinating open distributed systems. In *Proceedings of International Workshop in Future Trends in Distributed Computing Systems'99*, 1999.

[CD99b] Juan-Carlos Cruz and Stéphane Ducasse. A group based approach for coordinating active objects. In *Proceedings of Coordination'99, LNCS 1594*, pages 355–371, 1999.

[DDN00a] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. Finding refactorings via change metrics. In *Proceedings of OOPSLA'2000, ACM SIGPLAN Notices*, pages 166–178, 2000.

[DDN00b] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. Tie code and questions: a reengineering pattern. In *Proceedings of Europlop'2000*, 2000.

[DDN00c] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. Transform conditional: a reengineering pattern language. In *Proceedings of Europlop'2000*, 2000.

[DHN00] Stéphane Ducasse, Thomas Hofmann, and Oscar Nierstrasz. Openspaces: An object-oriented framework for reconfigurable coordination spaces. In António Porto and Gruia-Catalin Roman, editors, *Coordination Languages and Models*, LNCS 1906, pages 1–19, Limassol, Cyprus, September 2000.

[DLS00] Stéphane Ducasse, Michele Lanza, and Lucas Steiger. A query-based approach to support software evolution. In *ECOOP'2000 International Workshop of Architecture Evolution*, 2000.

[DLT00] Stéphane Ducasse, Michele Lanza, and Sander Tichelaar. Moose: an extensible language-independent environment for reengineering object-oriented systems. 2000. COSET'2000 (International Symposium on Constructing Software Engineering Tools).

[Duc99] Stéphane Ducasse. Evaluating message passing control techniques in smalltalk. *Journal of Object-Oriented Programming (JOOP)*, 12(6):39–44, June 1999.

[Küh98] Daniel Kühni. APROCO: A programmable coordination medium. Diploma thesis, University of Bern, October 1998.

[Lan99] Michele Lanza. Combining metrics and graphs for object oriented reverse engineering. Diploma thesis, University of Bern, October 1999.

[LAN00] Markus Lumpe, Franz Achermann, and Oscar Nierstrasz. A Formal Language for Composition. In Gary Leavens and Murali Sitaraman, editors, *Foundations of Component Based Systems*, pages 69–90. Cambridge University Press, 2000.

[Lum99] Markus Lumpe. *A Pi-Calculus Based Approach to Software Composition*. Ph.D. thesis, University of Bern, Institute of Computer Science and Applied Mathematics, January 1999.

[NA00] Oscar Nierstrasz and Franz Achermann. Separation of concerns through unification of concepts. In *ECOOP 2000 Workshop on Aspects & Dimensions of Concerns*, 2000.

[NSA00] Oscar Nierstrasz, Jean-Guy Schneider, and Franz Achermann. Agents everywhere, all the time. In *ECOOP 2000 Workshops on Component-Oriented Programming and Pervasive Component Systems*, 2000.

[Sch99] Jean-Guy Schneider. *Components, Scripts, and Glue: A conceptual framework for software composition*. Ph.D. thesis, University of Bern, Institute of Computer Science and Applied Mathematics, October 1999.

[SL00] Jean-Guy Schneider and Markus Lumpe. A Metamodel for Concurrent, Object-based Programming. In Christophe Dony and Houari A. Sahraoui, editors, *Proceedings of Langages et Modèles à Objets '00*, pages 149–165, Mont Saint-Hilaire, Québec, January 2000. Hermes.

[SN99] Jean-Guy Schneider and Oscar Nierstrasz. Components, scripts and glue. In Leonor Barroca, Jon Hall, and Patrick Hall, editors, *Software Architectures – Advances and Applications*, pages 13–25. Springer, 1999.

[TCD00] Sander Tichelaar, Juan Carlos Cruz, and Serge Demeyer. Design guidelines for coordination components. In Janice Carroll, Ernesto Damiani, Hisham Haddad, and Dave Oppenheim, editors, *Proceedings ACM SAC 2000*, pages 270–277. ACM, March 2000.

[TDD00] Sander Tichelaar, Stéphane Ducasse, and Serge Demeyer. FAMIX: Exchange experiences with CDIF and XMI. In *Proceedings of the ICSE 2000 Workshop on Standard Exchange Format (WoSEF 2000)*, June 2000.

FAMOOS Esprit Project Related Publications

[DD99] Serge Demeyer and Stéphane Ducasse. Metrics, do they really help? In Jacques Malenfant, editor, *Proceedings LMO'99 (Languages et*

Modèles à Objets), pages 69–82. HERMES Science Publications, Paris, 1999.

[DDL99] Serge Demeyer, Stéphane Ducasse, and Michele Lanza. A hybrid reverse engineering platform combining metrics and program visualization. In Francoise Balmas, Mike Blaha, and Spencer Rugaber, editors, *WCRE'99 Proceedings (6th Working Conference on Reverse Engineering)*. IEEE, October 1999.

[DDT99a] Serge Demeyer, Stéphane Ducasse, and Sander Tichelaar. A pattern language for reverse engineering. In Paul Dyson, editor, *Proceedings of the 4th European Conference on Pattern Languages of Programming and Computing, 1999*, Konstanz, Germany, July 1999. UVK Universitätsverlag Konstanz GmbH.

[DDT99b] Serge Demeyer, Stéphane Ducasse, and Sander Tichelaar. Why unified is not universal. UML shortcomings for coping with round-trip engineering. In Bernhard Rumpe, editor, *Proceedings UML'99 (The Second International Conference on The Unified Modeling Language)*, LNCS 1723, Kaiserslautern, Germany, October 1999. Springer-Verlag.

[Dem99] Serge Demeyer. Structural computing: The case for reengineering tools. In Peter Nuernberg, editor, *Proceedings of the 1rst Workshop on Structural Computing – Hypertext'99*, February 1999.

[DRD99] Stéphane Ducasse, Matthias Rieger, and Serge Demeyer. A language independent approach for detecting duplicated code. In Hongji Yang and Lee White, editors, *Proceedings ICSM'99 (International Conference on Software Maintenance)*, pages 109–118. IEEE, September 1999.

[DRG99] Stéphane Ducasse, Matthias Rieger, and Georges Golomingi. Tool support for refactoring duplicated OO code. In Stéphane Ducasse and Oliver Ciupke, editors, *Proceedings of the ECOOP'99 Workshop on Experiences in Object-Oriented Re-Engineering*. Forschungszentrum Informatik, Karlsruhe, June 1999. FZI-Report 2-6-6/99.

[DRN99] Stéphane Ducasse, Tamar Richner, and Robb Nebbe. Type-check elimination: Two object-oriented reengineering patterns. In Francoise Balmas, Mike Blaha, and Spencer Rugaber, editors, *WCRE'99 Proceedings (6th Working Conference on Reverse Engineering)*. IEEE, October 1999.

[RD99] Tamar Richner and Stéphane Ducasse. Recovering high-level views of object-oriented applications from static and dynamic information. In Hongji Yang and Lee White, editors, *Proceedings ICSM'99 (International Conference on Software Maintenance)*, pages 13–22. IEEE, September 1999.

[Ric99] Tamar Richner. Using recovered views to track architectural evolution. In *ECOOP'99 Workshop Reader*, number 1743 in LNCS. Springer-Verlag, June 1999.

[TD99] Sander Tichelaar and Serge Demeyer. SNiFF+ talks to Rational Rose – interoperability using a common exchange model. In *SNiFF+ User's Conference*, January 1999. Also appeared in the "Proceedings of the ESEC/FSE'99 Workshop on Object-Oriented Re-engineering (WOOR'99)" – Technical Report of the Technical University of Vienna (TUV-1841-99-13).

Other Publications

[BDG99] Isabelle Borne, Serge Demeyer, and Galal Hassan Galal. Proceedings of the ECOOP'99 workshop on object-oriented architectural evolution, June 1999.

[DG99] Serge Demeyer and Harald Gall, editors. *Proceedings of the ESEC/FSE'99 Workshop on Object-Oriented Re-engineering (WOOR'99)*. TUV-1841-99-13. Technical University of Vienna - Information Systems Institute - Distributed Systems Group, September 1999.

[DG00] Serge Demeyer and Harald Gall. Workshop on object-oriented reengineering (WOOR'99). *Software Engineering Notes*, 25(1), January 2000.

[MD99] Ana Moreira and Serge Demeyer, editors. *Object-Oriented Technology (ECOOP'99 Workshop Reader)*. Number 1743 in LNCS. Springer-Verlag, Kaiserslautern, Germany, December 1999.

[NL99] Oscar Nierstrasz and Michel Lemoine, editors. *Proceedings ESEC/FSE'99*. LNCS 1687. Springer-Verlag, Toulouse, France, September 1999.