Annual Merit Report for 1998

Karl Lieberherr

March 24, 1999

1 Research

Our research moved towards applying our expertise in Adaptive Programming (AP) and Aspect-Oriented Programming (AOP) to component-based software development. This is a natural direction for the Demeter research group because of the current interest in component-based software development. It is a continuation of our earlier work because Ian Holland's PhD thesis on executable contracts [Hol93] completed in our group six years ago is an early work on components. The most important work of this year is our OOPSLA '98 paper with Mira Mezini as first author on Adaptive Plug-and-Play Components.

1.1 Adaptive Plug-and-Play Components (AP&P components)

AP&P components are an integration of Mira Mezini's PhD work on variation-oriented programming (Rondo) with Demeter ideas. See Mira Mezini's Ph.D. thesis [Mez97] for her related ideas on Rondo. Basically, an AP&P component is a piece of behavior that is formulated in an ideal context (an abstract class graph) and later mapped into a concrete context (a concrete class graph). The motivation for using an abstract class graph comes from our experience of programming against concrete class graphs. Such programs are both more brittle and longer than they need to be.

For expressing the mapping from abstract to concrete class graphs we can reuse our work on AP. It turns out that with AP we can express a mapping from an abstract class graph to a concrete class graph without committing to the details of the concrete class graph. This is a significant benefit because the concrete class graphs often change and maintaining the map can be time consuming. AP facilitates initial development and maintenance of the maps.

An AP&P component has a required and a provided interface and contains behavior definitions for the participants of the AP&P component. Two or more AP&P components may be connected by connectors that link the required and provided interfaces. The class graphs mentioned above are parts of those interfaces: the abstract class graph sits in a required interface and the concrete class graph in a provided interface.

Our earlier work on traversal/visitor style programming was the predecessor of AP&P components. In traversal/visitor style programming you separate traversal programming from programming of actions that need to be executed during traversal. This programming style is still useful with AP&P components. Because the AP&P components are written modulo an abstract class graph the use of traversal strategies to express navigation is less urgent. In other words, programming with AP&P components can be done by delegating all adaptiveness to the connectors only. This is a good approach for getting AP&P components used by practitioners because they can be implemented with pure oo techniques and the "clever" AP techniques are only used in the connectors that are conceptually easier than the AP&P components.

Programming with AP&P components is a form of parameterized programming where formal and actual parameters are linked by connectors and where the formal parameters define constraints on graphs that must be satisfied by the graphs serving as actual parameters. This is different than template programming in C++.

Next we define the concept of refinement-instance that we have developed over the last few years and that is fundamental to the concept of instantiating AP&P components. It needs to be stressed that the concept is independent of AP and is useful in contexts where graphs are "instantiated". We first introduce the concept of instance of a graph. The graphs we refer to here are UML class diagrams; however, to understand the definition of refinement-instance it is sufficient to consider graphs with labeled nodes and labeled edges without distinguishing between different kinds of nodes and edges. This generalization is not hard and is presented in papers on AP [PXL95] and environmental acquisition [GL96]. Informally, a graph G is an instance of graph S if G contains the connectivity of S. We think of S as an abstract graph and G as a concrete graph. Formally, a graph G is an instance of a graph S, if S is a connected subgraph of the transitive closure of G.

Informally, a graph G is a refinement-instance of a graph S if the connectivity of S is in "pure form" in G. By pure form we mean that there are no "surprise uses" of the nodes of S. Formally, a graph G is a refinement-instance of a graph S, if S is a connected subgraph of the pure transitive closure of G with respect to the node set of S. The pure transitive closure of G=(V,E) with respect to a subset W of V is the graph G*=(V,E*), where E*=(i,j): there is a W-pure path from vertex i to vertex j in G. A W-pure path from i to j is a path where i and j are in W and none of the inner points of the path are in W.

Connectors use traversal specifications to specify class graph to class graph mappings. Traversal specifications are best thought of as anchored graph patterns which match a substructure of the class graph or a substructure of an object graph. The graph pattern can be thought of as a regular expression over the alphabet of the node and edge labels of the class graph. A special symbol "any" is used to denote any node or edge in the class graph. For example, A any* B, denotes the set of paths starting at node A and going through several edges and terminating in node B. Instead of using a regular expression notation for traversal specifications, we prefer to use strategy graphs. A strategy graph is a directed graph which defines positively the overall topology of the traversal. The edges are labeled with constraints which describe negatively the nodes and edges in the class graph which must be avoided. We like such a separation into positive and negative information although it would also be possible to assign regular expressions with "any" to the edges of a strategy graph. In such a representation the regular expressions on the edges could express further positive information about the traversal.

The compilation of strategy graphs posed interesting problems. Palsberg, Xiao, Lieberherr [PXL95] presented a fast compilation algorithm for a restricted set of propagation directive and class graph combinations. Palsberg, Patt-Shamir, Lieberherr [PPL97] present a general compilation algorithm PPL which may be exponential for some inputs. Indeed, Lieberherr, Patt-Shamir [LPS97] show that the PPL algorithm is exponential for certain inputs. Lieberherr, Patt-Shamir [LPS97] also show a polynomial general compilation algorithm LP for any combination of strategy graphs and class graphs. Algorithm LP can be roughly summarized as follows. The summary is rough because it does not deal with abstract classes and inheritance. The key subcomputation is the intersection of non-deterministic finite automata. The strategy graph is viewed as a non-deterministic finite automaton and the intersection with the class graph NDFA yields a new automaton, called a traversal graph automaton. The application of the traversal graph automaton to a specific object graph can again be viewed as an NDFA intersection problem, namely the intersection of the traversal graph NDFA and the object graph DFA. In practice, the traversal of an object graph according to a traversal graph automaton is done in a similar way as the simulation of an NDFA. Algorithm LP solves the compilation problem in a very satisfactory way.

Will Clinger and Lars Hansen and Mitch Wand observed that strategy graphs only allow for a certain kind of normalized traversals and that for some applications the traversals should be more complex. Mitch Wand and Johan Ovlinger [OW98] proposed that we use traversal automata to specify traversals of object graphs. Traversal graphs encode the details of the object structure in them and are therefore not structure-shy. Strategic traversal automata were then invented as a way to describe more complex traversals in a structure-shy way. Instead of being formulated for a concrete class graph, strategic traversal automata are formulated in terms of an abstract graph.

1.2 Class Graph Views

Johan Ovlinger developed the concept of class graph views. Views are closely related to AP&P components, in that views are the tools with which programming constructs like AP&P components can be implemented.

An AP&P component is a high-level programming construct for promoting code reuse. It provides the programmer with tools to write code that is easily added to a variety of host class graphs. AP&P components are a vehicle for research into programming constructs for reusable code.

AP&P components are implemented by using class graph views. A view is a low-level programming pattern that is able to view one class graph as though it were another. Views are a vehicle for research into implementations of module systems for object oriented languages.

Class graph views also incorporate some of the goals of Subject Oriented Programming [HO93], but at a lower level than originally proposed. Class graph views thus incorporate the core features of reuse, module systems, and subject oriented programming into a single programming pattern.

The following terminology/capability maps show similarities and differences (APPC = AP&P component):

```
Similarities
=====================================================
Views                   APPC

view classes            participant classes

provided interface      provided interface

required interface      required interface

map                     connector

view class graph        interface class graph

host class graph        implementation class graph

apply one view to       apply one APPC to
several host class      several implementation class
graphs                  graphs

apply one view          apply one APPC
several times to        several times to same
same host class graph   implementation class graph

apply multiple views    apply multiple APPC
```

```
to same host           to same implementation
class graph            class graph

compatibility between  compatability between
view class graph and   interface class graph and
host class graph       implementation class graph

views are reusable     APPC are reusable
components             components

Differences
=====================================================
Views                  APPC

host-to-view           view-to-host
layering               layering

instantiate view       participant classes
classes                are not instantiated

supports basics of     not the case because
subject-oriented       APPC are not
programming            self-contained

intent:                intent:
modularization         reuse
unintentional reuse    intentional reuse

hide host behind view  inject view into host

view clients provided  APPC clients provided
with higher            with code
abstraction
```

The goal of both Views and AP&P components is reuse. Other differences between Views and AP&P components are: Views are intended to be more basic. Views only use Java code in the maps while AP&P components may use higher-level constructs such as traversal strategies and regular expressions in the connectors.

Class graph views are similar to Catalysis type models [DW98]. They are abstractions of a UML class model in a similar way as interface class graphs are abstractions of concrete class graphs. Unlike a UML class diagram that describes the design for a single implementation, a type model captures constraints that must be true in any conforming implementation. In Catalysis, the behavior of type models is defined by type collaborations that are an abstraction of the UML collaboration concept.

The connection between Catalysis and Views and AP&P components needs to be further studied. AP&P components are different than Catalysis type models because of the adaptive features.

1.3 Personalities

Luis Blando developed the concept of personalities. A personality is basically a Java interface with code or an AP&P component for one class only. A personality describes popular behaviors of classes. A personality may be used by several personifying classes. See [Bla98, BLM99] and

`http://www.ccs.neu.edu/home/lblando/personalities/`

for more information. An implementation was developed in Argentina under Luis Blando's direction.

1.4 Demeter/Java

Demeter/Java was in 1998 further refined by Doug Orleans. See:

`http://www.ccs.neu.edu/research/demeter/DemeterJava/`

(especially the CHANGES file).

Demeter/Java is our main tool to develop adaptive programs. Demeter/Java was used very successfully in COM 3360 in the fall of 1998 and in COM 3362 in the spring of 1998.

Demeter/Java enjoys an active user group. We don't know its size but we expect to have about 100 active users outside Northeastern. Traversal/visitor style programming, the concept behind Demeter/Java, is probably much more widely used.

1.5 AP Library

In 1998, Doug Orleans extracted a separate reusable library from Demeter/Java. It is called the the AP library and contains the core algorithms for compiling adaptive programs. The AP library is currently used in both Demeter/Java and DJ (see below). For more information, see:

`http://www.ccs.neu.edu/research/demeter/AP-Library/`

1.6 Aspects

In 1998, Josh Marshall has perfected his generic weaver that is used by Demeter/Java. The generic weaver can be viewed as a precursor to the AP&P component language for AOP. The weaver is an auxiliary meta program which is used for implementing aspects in general, not just COOL and RIDL.

Johan Ovlinger implemented RIDL for Demeter/Java.

We developed a definition for AOP, based on earlier work in [**?**], that is better than the Xerox PARC AOP definition. Our definition is not programming language dependent and is applicable also to aspect-oriented information engineering, not just AOP. See:

`http://www.ccs.neu.edu/home/lieber/AOP.html`

1.7 Visual Programming

Kedar Patankar and Binoy Samuel have refined AP Studio for visual adaptive programming. AP Studio allows you to develop strategy graphs visually in the context of a concrete class graph and to visualize the selected path set as a subgraph. The UML class diagram notation is used to represent class graphs. Doug Orleans has integrated AP Studio into Demeter/Java.

1.8 DJ

DJ is the little sister of Demeter/Java described at

`http://www.ccs.neu.edu/research/demeter/DJ`

and implemented by Joshua Marshall. With DJ you can do AP without having to learn a new language. All programming is done in Java by calling a few methods from a library. We hope that DJ will eventually lead to a much larger dissemination of AP ideas.

1.9 Books in Print

Karl J. Lieberherr, "Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns", *Book published by PWS Publishing Company*, 1996, 650 pages.

1.10 Journal Articles (Published)

"Evolution of Object Behavior Using Context Relations", with Linda Seiter and Jens Palsberg, *IEEE Transactions on Software Engineering*, Vol. 24, no. 1, pages 79-92, 1998.

1.11 Journal Articles (Submitted)

"Generic Programming with Graph Refinement", with Boaz Patt-Shamir, *Schloss Dagstuhl Proceedings on Generic Programming*, 1998.

`http://www.ccs.neu.edu/research/demeter/biblio/graph-refine.html`

1.12 Refereed Conference Papers Published

"Adaptive Plug-and-Play Components for Evolutionary Software Development" with Mira Mezini. OOPSLA '98. *1998 Proc. Object-Oriented Programming Systems, Languages and Applications Conference (OOPSLA), Special Issue of SIGPLAN Notices*, Vol. 33, no. 10, pages 97-116, ACM Press.

An anonymous OOPSLA '98 reviewer evaluates and describes AP&P components as follows: *This is the kind of work that the software industry needs most. Hard core work, aimed at developing tangible and implementable mechanisms to help bridge the chasm between illusive design methodologies of arrows, bubbles, use cases, processes, etc., and the tough reality of the programmers in the trenches. Even the simple observation and unequivocal statement that a design segment should be a distinct lingual construct, have formal parameters that can be matched against several locations in the class graph merits honorable publication, since most previous work tried and failed to do so while using existing programming constructs (see e.g., observer pattern implementations using inheritance and templates). There is however in the paper much more than that, ...*

"From CSCW Applications to Multicast Routing: An Integrated QoS Architecture" with I. Matta and M. Eltoweissy. *1998 Proc. IEEE International Conference on Communications–ICC*, Atlanta, Georgia.

Give Ibrahim 90% for this paper and give me 10%.

1.13 Refereed Conference Papers Accepted

"Modeling Behavior with Personalities" with Luis Blando and and Mira Mezini, *1999 International Conference on Software and Knowledge Engineering*, Kaiserslautern, Germany, 7 pages.

1.14 Unrefereed Publications

The Demeter home page has numerous links to unrefereed publications. They attract a significant readership based on the web statistics.

1.15 Invited Talks, Colloquia, etc.

DARPA PI meeting, Los Angeles, CA, March 16 - March 19

Schloss Dagstuhl Workshop on Generic Programming (April 1998, 1 week)

July 19 - July 24: DARPA EDCS Demo Days and PI meeting in Baltimore. Demonstration of Demeter/Java during 2 days with Doug Orleans.

1.16 Technology Transfer Activities

We (with Jens Palsberg and Boaz Patt-Shamir) have applied for a US Patent for Compiling Adaptive Programs. I had numerous discussions with the patent lawyers and the patent examiner and finally the patent was approved. It will be formally issued in 1999. It is based on [PPL97, LPS97].

Technology Transfer Activities is a very significant effort for us because of our involvement with DARPA which asks for technology transfer reports frequently. Our effort goes into developing, maintaining and promoting Demeter/Java, and the AP Library and into promoting the ideas behind AP. We use the Demeter home page

`http://www.ccs.neu.edu/research/demeter/`

as a very important technology transfer tool and spend significant time in maintaining those pages.

I am on the technical advisory board for Tendril Software, Inc. They develop a tool using some of the Demeter ideas.

There are projects using our technology at companies such as Xerox, GTE, Motorola, HP, Novell and SUN.

1.17 Grants

1. ARPA EDCS grant, $750000 1996-1999, Karl Lieberherr, PI, Evolution of Software via Adaptive Programming, DARPA and Rome Laboratory under agreement F30602-96-0239.

1.18 Grant Proposals

I wrote two grant proposals to DARPA with outside people. One was with GTE/BBN (Zincky/Anderson), the other with Miriam Leeser and David Lorenz. Unfortunately, both were not funded.

1.19 Conferences Attended

ECOOP PC meeting, January 1998.

Oct. 18 - 22: OOPSLA '98, Vancouver, Canada.

Two DARPA PI meetings mentioned earlier.

2 Teaching

2.1 Courses

I taught three courses becuase of a DARPA buyout. I developed a new course on software testing (COM 3220, 7 students).

`http://www.ccs.neu.edu/home/lieber/com3220/com3220.html`

I taught my favorite course, COM 3360.

`http://www.ccs.neu.edu/home/lieber/com3360.html`

Maybe it should count as 1.25 courses since I teach two sections of the course: the local section (18 students) and the national NTU section (3 students). I prepared many hundred new viewgraphs for this course which are all available from my home page in Postscript and PowerPoint formats. The course went very well this Fall. Many successful Demeter/Java projects were completed.

I also taught COM3362, Advanced Object-Oriented Systems, with 6 students. It is another one of my favorite courses.

`http://www.ccs.neu.edu/home/lieber/com3362/com3362.html`

2.2 Curriculum Development

I developed a new course on software testing (see above).

2.3 Seminar

Preparation for the Demeter Seminar takes a significant amount of time since it requires reading the latest literature relevant to our project. There is quite a bit of that. For many of the seminars I prepare PowerPoint viewgraphs which are available from my home page.

2.4 Ph.D. Students

Currently, Doug Orleans and Johan Ovlinger are my Ph.D. students. They both are working towards their thesis proposal.

2.5 Ph.D. Committees

I serve on the committee of Michael Werner. He comes regularly to the Demeter seminar and is working on topics of interest to our research group. The advisor is Ken.

2.6 Master's Thesis

Luis Blando has written his Master's thesis with Mira Mezini and me as advisors. See

`http://www.ccs.neu.edu/home/lblando/personalities`

2.7 Reading and Project Courses

I supervised reading courses by my Ph.D students Doug Orleans and Johan Ovlinger.

3 Service

3.1 Summary

A major service component is the Editor-in-chief position for TAPOS (Theory and Practice of Object Systems, a John Wiley journal).

3.2 College Committees

I am on the Graduate Committee.

I am on the hiring committee. I took an active role in attracting both Mira Mezini and David Lorenz to our College.

I am on the tenure committee. No activity.

I am on the full professor committee. No activity.

I am on the sabbatical committee (chair).

With Mitch and Will I am on the PL languages field committee.

3.3 University Committees

I am a member of the University Graduate Council. I am on the program review committee that started meetings again in the fall.

3.4 Professional Service

Editor-in-chief for TAPOS. Solicite and review papers and select about 16 per year (4 issues).

I was on the program committee of ECOOP '98 and ECOOP '99 reviewing about 25 papers in the December/January time frame. I attended the the 1998 PC meeting in Copenhagen (January 98) and the 1998 PC meeting in Lausanne (January 99).

References

[Bla98] Luis Blando. Designing and Programming with Personalities. Technical Report NU-CCS-98-12, Northeastern University, December 1998. Master's Thesis.

[BLM99] Luis Blando, Karl Lieberherr, and Mira Mezini. Modeling Behavior with Personalities. In *International Conference on Knowledge and Software Engineering*, Kaiserslautern, 1999.

[DW98] D.F. D'Souza and A.C. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach.* Addison-Wesley, 1998.

[GL96] Joseph Gil and David H. Lorenz. Environmental Acquisition–A new inheritance-like abstraction mechanism. In *Proceedings of the 11th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 214–231, San Jose, California, October 6-10 1996. OOPSLA'96, Acm SIGPLAN Notices 31(10) October 1996.

[HO93] William Harrison and Harold Ossher. Subject-oriented programming (A critique of pure objects). In *Proceedings OOPSLA '93, ACM SIGPLAN Notices*, pages 411–428, October 1993. Published as Proceedings OOPSLA '93, ACM SIGPLAN Notices, volume 28, number 10.

[Hol93] Ian M. Holland. *The Design and Representation of Object-Oriented Components.* PhD thesis, Northeastern University, 1993.

[LPS97] Karl J. Lieberherr and Boaz Patt-Shamir. Traversals of Object Structures: Specification and Efficient Implementation. Technical Report NU-CCS-97-15, College of Computer Science, Northeastern University, Boston, MA, Sep. 1997.

[Mez97] Mira Mezini. *Variation-Oriented Programming Beyond Classes and Inheritance.* PhD thesis, University of Siegen, 1997.

[OW98] Johan Ovlinger and Mitchell Wand. A Language for Specifying Traversals of Object Structures. Technical Report NU-CCS-98-

??, College of Computer Science, Northeastern University, Boston, MA, November 1998.

[PPL97] Jens Palsberg, Boaz Patt-Shamir, and Karl Lieberherr. A new approach to compiling adaptive programs. *Science of Computer Programming*, 29(3):303–326, 1997.

[PXL95] Jens Palsberg, Cun Xiao, and Karl Lieberherr. Efficient implementation of adaptive software. *ACM Transactions on Programming Languages and Systems*, 17(2):264–292, March 1995.