

Project Summary

Active Information Delivery Using Selector Languages

Karl Lieberherr
Ravi Sundaram

We address the issue of active information delivery during the programming of "Have X, Want Y" concerns. Such programming concerns are very common and typically cut across multiple types whose signatures may be unfamiliar to the programmer. In active information delivery systems ([?]), the relevant information is delivered at the right moment for the programmer to consider. Applied to "Have X, Want Y" concerns this means that we compute the set of all paths from X to Y in the signature graph and let the programmer choose the correct path or set of paths.

PROJECT DESCRIPTION

We propose to develop a theory of selector languages and a tool, called Demeter/P, that has the goal to help programmers to more productively develop applications reusing libraries. The problem today is that these libraries are becoming too large in size making their usage and understanding difficult and time consuming. We address the problem on two fronts: First we make it easier to identify the relevant parts of a library needed to achieve a specific task and second we support evolution of the code should the library change. The programming tasks that we consider are of a very common form: "Have X, want Y", i.e., given an object of type X, we want to retrieve or construct an object of type Y with the assumption that the given library contains enough primitive methods so that Y can be constructed by calling those primitive methods and enhancing them with zero or more before/after or around methods. We will also consider the generalization: "Have X_1, \dots, X_n , want Y".

A theory of selector languages is needed as a theoretical foundation of Demeter/P like tools. A selector language operates in the space of graphs, that we call meta graphs, and instances of those graphs that are themselves graphs, called instance graphs. A selector language expression selects a set of nodes or edges for a given instance graph that conforms to some meta graph. When using selector languages in practical tools, the following algorithmic properties should be checked: Satisfiability, Implication, Always, Never, etc. ([?]) and we need algorithms for learning and optimizing selector expressions.

Jeff's example.

The benefits of Demeter/P are:

- The tool identifies the parts of the library that is relevant to a given task. This facilitates task-specific learning of the library.
- Demeter/P proposes solutions to "Have X, want Y" problems at an appropriate level of abstraction revealing the intent why we want the Y in the form of a simple query. This query is used to evolve the software.
- When the library evolves, the solutions to the "Have X, want Y" problems might change. Demeter/P helps with the evolution by generalizing appropriately and by notifying the programmer where a manual intervention is needed.

Demeter/P operates as follows. The user formulates an initial query, "Have X, want Y", and the tool responds with a set of queries that selects the k shortest path (k=5) and all paths.

The graph in which the paths are computed is the class graph of the library. Initially, the class graph will only contain 0-argument methods and static methods with one argument.

The initial prototype will use the DJ library that already builds the class graph for 0-argument methods.

How should the DJ and AP libraries be expanded?

Add a qualifier: shortest: { A -> B B -> C } does not find all paths satisfying the constraint, only the shortest satisfying the constraint.

The algorithm for computing the shortest path satisfying the constraint first computes the traversal graph followed by choosing the shortest path in the traversal graph.

Refine query after the results of the simple query have been seen. Initial query:

[IFile,ASTNode]

After seeing the results and studying the documentation of ICompilationUnit:

[IFile,ICompilationUnit] [ICompilationUnit, ASTNode]

How to deal with arguments? Choose constants for primitive types.

For Dan and Keith:

Read Prospector paper Read Persephone paper paper with Fabio

get Prospector Eclipse plugin from Berkeley and install it.

Try Prospector on the Busroute library. Have a BusRoute, want a Person. Does Prospector do the right thing?

Enhance Prospector so that for the BusRoute example:

from BusRoute to Person

should produce: (the equivalent at the Java level)

from BusRoute via BusStop to Person from BusRoute via Bus to Person from BusRoute to Person

Scientific contributions:

have X want Y also in other contexts: Have call to main(), want call to foo(). Navigation on execution tree.

Learn query from meta graph paths.

selector expression minimization, learning, type checking, compilation.

1 Proposed Research

We propose to continue our research on selector languages. Specifically, we would like to

1. continue our investigation of the connection between selector languages and language constructs;
2. find out how to build selector expressions interactively;
3. study the use of selector expressions for software evolution;
4. study the theory of selector languages, both algorithmic lower and upper bounds for checking important properties of selector expressions, such as Satisfiability, Implication, Always, Never;
5. develop a reusable framework for the implementation of selector languages;
6. use the reusable framework for building the programming tool Prospector/P.

The following sections explain these goals in detail.

1.1 The Essence of Selector Languages

2 Time Line and Management Plan

3 Significant Impact

3.1 Social

3.2 Education

3.3 PhD Student Team

3.4 Results from Prior NSF Support for Karl Lieberherr

We briefly summarize the results from 4 previous NSF grants. Our work was influential in shaping the area of Aspect-Oriented Software Development that has now its own conference series.

- NSF-Grant CCR-0098643, with David Lorenz (2001-2002).
Title: Collaboration-oriented Aspects
We improve previously exponential algorithms for compiling aspect-oriented programs to become polynomial time [?]. [?] shows a good way to combine aspects and modules. In [?] we show how to use AspectJ to check the Law of Demeter and outline how to improve AspectJ's static checking capabilities. In [?] we continue our earlier work on domain-specific aspect languages and provide a tool that we want to apply to apply to cybertrust. [?] summarizes our ideas for better referential mechanisms in programming languages.
- NSF-Grant CCR-9402486 (Software Engineering) (1994-97). Cosponsored by DARPA.
Title: Engineering Adaptive Software
The key results of this grant are a formalization of the important concepts of adaptive programming [?], fast compilation algorithms [? ?], an evolution framework for adaptive programming [? ?], an application of adaptiveness to parameter passing in distributed systems [?], and the distribution of the Adaptive Programming tools through a undergraduate/graduate level textbook [?].
- NSF-Grant CCR-9102578 (1991-93).
Title: Abstractions for Organizing Classes
The key result of this grant is a new method to develop software based on separating the concerns: structure, traversal and traversal-based collaboration. Publications: [? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?].
- NSF-Grant MCS80-04490 (1980-1982), NSF-Grant MCS82-01878 (1982-1983).
Title: Combinatorial Optimization and Search Problems
The key contributions of this work are: 1. A seminal contribution to the state-of-the-art in hardware description languages which resulted in the invited paper [?]. 2. A theory of P-optimal approximation algorithms for combinatorial optimization problems. Publications: [? ? ? ? ? ? ? ? ? ?].

May 17, 2005
Draft Version

BUDGET JUSTIFICATION