

# Query-based Modeling and Fluid

Elissa Newman  
Carnegie Mellon University  
5000 Forbes Ave.  
Pittsburgh, PA 15213  
elissa@cs.cmu.edu

## 1. INTRODUCTION

*Query-based modeling* is an iterative process of querying code, obtaining a response, visualizing the results, and eventually recording design intent as query/response pairs. *Query-based constraints* are the query/response pairs that were introduced in our previous paper [5]. Query-based constraints are used to assure that the ad hoc design intent as expressed in the query/response pair remain true during code evolution. The recorded intent also serves as low-level documentation of the code.

The programmer trades off his time and mental effort in expressing the constraints in return for improved evolvability and understandability of the design intent of the program. The programmer also receives an immediate benefit: ongoing assurance that the design intent expressed is consistent with the code. Design intent is a level of design information closer to the as-built code than traditional design documentation. In Fluid, design intent is expressed as annotations in the code.

The Fluid project [2] combines multiple static software analyses that focus on mechanical attributes of programs. These include effects [3], unique references, aliasing [3], use of locks in multi-threaded programs [4], policy restrictions on subclassing [1], concurrency policy [4], and relationships among code segments, threads, and shared data [6]. This additional semantic information provides us the opportunity to provide semantic querying of code via our rich query language FQL (Fluid Query Language)<sup>1</sup>. FQL combines traditional query language features such as code structure, defs and uses, type hierarchies, and method calls with the information provided by our mechanical program analyses listed above.

The richness of our query language provides power to the constraint system. Any query that can be expressed in FQL can be used as the query part of a constraint.

## 2. EXAMPLES

### 2.1 Thread-local

**Query:** “Which state protected by a lock is touched by only one thread?”

**Response:** Empty set

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SPLAT '05 at AOSD '05, March 15, 2005, Chicago, Illinois, USA.  
Copyright 2005 ACM 1-58113-000-0/00/0004...\$5.00.

**Analyses Used:** Lock analysis (“which state is protected by a lock”) and Thread Color analysis (“is touched by only one thread”)

The programmer may wish to make this query/response pair a constraint on the code, as state that is only touched by a single thread is not shared state and should not be protected by a lock.

### 2.2 Deadlock

**Query 1:** “Which locks protect the data written in method foo in class C?”

**Response 1:** “Locks InstanceLock and ArrayLock”

**Analyses Used:** Lock analysis (“Which locks protect the data”) and Effects analysis (“data written in method foo in class C”)

**Query 2:** “What is the global lock ordering among the locks InstanceLock and ArrayLock?”

**Response 2:** “InstanceLock then ArrayLock”

**Analyses Used:** Lock order analysis

**Query 3:** “Where could ArrayLock ever be acquired without first holding InstanceLock?”

**Response 3:** “In method baz in class C”

**Analyses Used:** Lock analysis

The programmer would then want to check the ordering of the lock acquisitions in method baz to see if either baz does not acquire InstanceLock at all or if it acquires InstanceLock after acquiring ArrayLock. Either of these alternatives could cause deadlock. If the true intent was to acquire InstanceLock then ArrayLock, the programmer could then ensure that this intent remain true by making it a constraint.

### 2.3 Hierarchy and Region Aggregation

**Query 1:** “Which classes other than C may touch this package private array?”

**Response 1:** “Subclass D and class F”

**Analyses Used:** Defs/Uses

**Query 2:** “Does class D aggregate the array’s elements into a region?”

**Response 2:** “Yes, into region ArrayState”

**Analyses Used:** Lock analysis

**Query 3:** “Which lock protects region ArrayState?”

**Response 3:** “Lock ArrayLock in class C”

**Analyses Used:** Lock analysis

These examples have shown how the iterative querying process and the query-based modeling process work, as well as how multiple analyses are involved in formulating the answers to the queries.

### 3. ASPECT-ORIENTED APPLICATIONS

Query-based modeling can also be applied to aspect-oriented technologies. Sets of queries can be combined into *concern manifestations* that provide boundaries for concerns in the code. As the code evolves, the Fluid tool will compute the new set of program elements in the concern based on the queries defined in the concern manifestation. The programmer may also wish to make the set of program elements currently in the concern manifestation into a constraint, so that changes to the scope of the concern can be detected and analyzed as the code evolves. Query-based modeling can also be used to check that the query-based constraints hold for base code as well as woven code. In these ways, query-based modeling can be used to check consistency.

### 4. CONCLUSION

In summary, query-based modeling trades programmer effort in making explicit the low-level design intent in the code base via

query-based constraints for increased evolvability, understandability, and consistency.

### 5. REFERENCES

- [1] E.C. Chan, J. T. Boyland, and W.L. Scherlis. *Promises: Limited Specifications for Analysis and Manipulation*. In ICSE'98.
- [2] Fluid webpage. <http://www.fluid.cs.cmu.edu>
- [3] A. Greenhouse and J. Boyland. *An Object-Oriented Effects System*. In ECOOP'99.
- [4] A. Greenhouse and W.L. Scherlis. *Assuring and Evolving Concurrent Programs: Annotations and Policy*. In ICSE 2002, 453-463, May 2002.
- [5] E. Newman and W.L. Scherlis. *Towards Query-based Constraints*. In SPLAT 2003 at AOSD 2003, March 2003.
- [6] D. F. Sutherland, A. Greenhouse, and W.L. Scherlis. *The Code of Many Colors: Relating Threads to Code and Shared State.* In Workshop on Program Analysis for Software Tools and Engineering (PASTE'02), at FSE-10, November 2002.

---

<sup>i</sup> FQL will be the subject of a future publication. We use structured English queries for the examples in this paper.