STAC: Automatically Identifying Software Tuning Parameters

Nevon Brake
James R. Cordy
School of Computing, Queen's University
Kingston, Ontario, Canada

Elizabeth Dancy
Marin Litoiu
Valentina Popescu
IBM, Toronto Lab
Markham, Ontario, Canada

I. Introduction

Software Tuning Panels for Autonomic Control (STAC)

- Automatically re-architects legacy source code for autonomic control
- Source transformation rules written in TXL generate a control panel to centralize access to and modification of tuning parameters
- XML mark-up is inserted manually to identify tuning parameter declarations

What are tuning parameters?

- Scalar fields and scalar properties of structured fields
- Explicit declarations in source code
- Used to influence or observe the behaviour of the system
- Related to metrics such as performance and security

Where are tuning parameters?

- Tuning parameters are not always documented, intentionally and unintentionally
- Variable names provide clues but can be misleading or ambiguous
- Not always explicit (e.g., cache hit rate, tree depth)

Objective

- Automate tuning parameter identification and mark-up using patterns of use

II. Case Studies

Creating a taxonomy

- Studied four server-oriented applications implemented in Java:
  - Apache Tomcat/Jetty (Web/Servlet)
  - Apache Derby/Berkeley DB Java Edition (Database)
- Catalogued tuning parameters from manuals, source comments, JMX
- Classified according to usage patterns (Fig. 1)

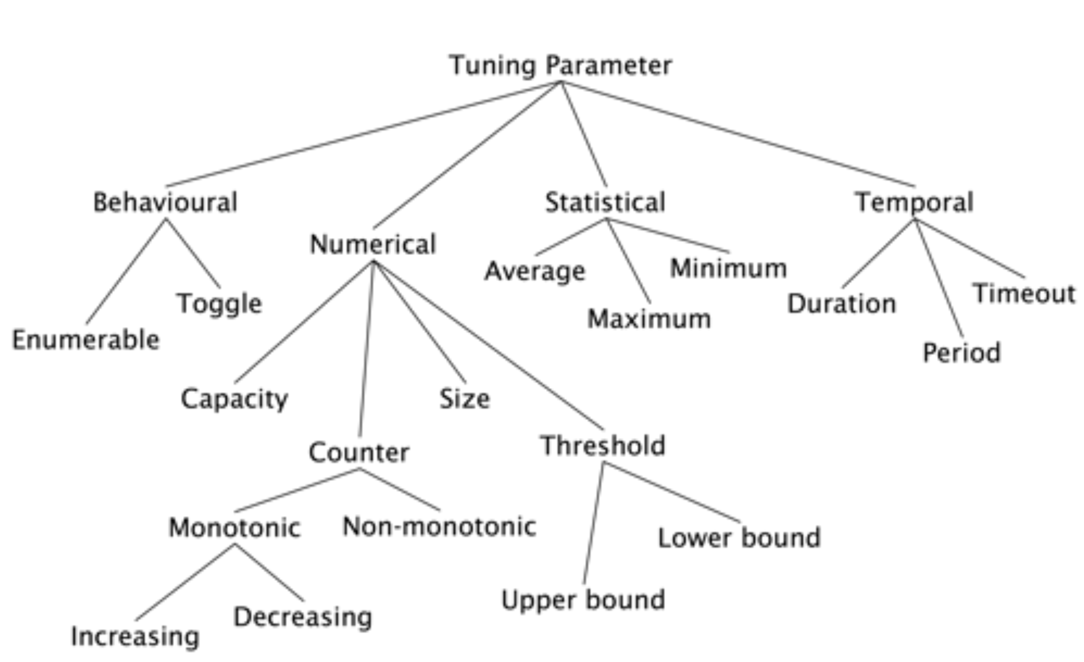

Figure 1. Tuning parameter classification

III. E-R Model

Building the model

- Need to know where and how data are transferred and how data are compared
- Entities abstracted from the Eclipse JDT program model to represent:
  - Types
  - Fields
  - Local variables
  - Methods
- Relationships represent data transfers and data comparisons between entities (Fig. 2)
- Model refined iteratively based on the demands of each usage pattern
- Relationships are sub-typed to provide stronger semantics

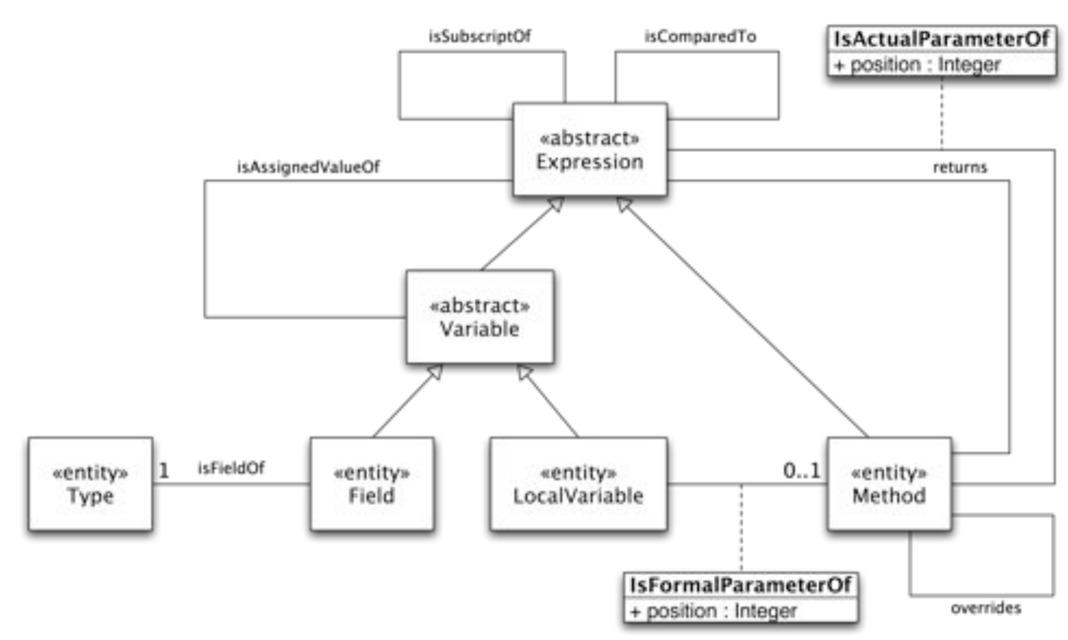

Figure 2. Entity-Relationship model

Implementing the model

- Limited to binary relationships only
- Each relationship class is represented as a set of 2-tuples
- Tuples are defined recursively to support composite entities and relationship attribution

IV. Fact Extraction

Abstract Syntax Trees

- Eclipse JDT parser used to generate AST for each compilation unit
- Nodes of AST visited to extract instances of relationships between entities
- Extracted relationships form a directed graph of tuples (Fig. 3)
- Graph patterns used to identify tuning parameters

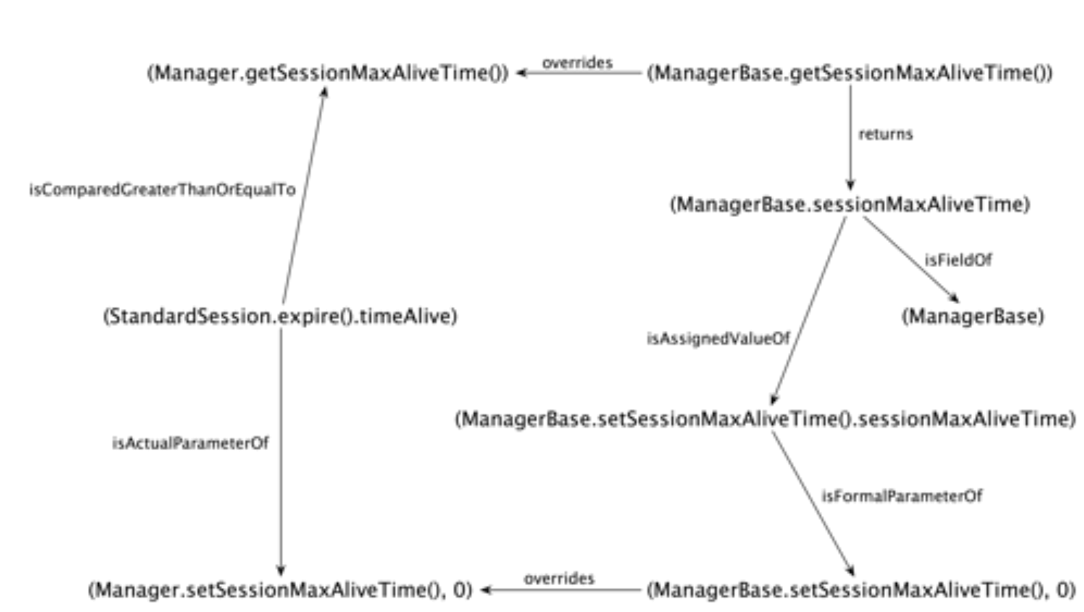

Figure 3. Facts extracted from several classes in Apache Tomcat

V. Graph Manipulation

Manipulating the facts

- Operations on binary relationships used to manipulate facts
- Isolated facts from each compilation unit are combined to elicit program understanding
- Inferences create new edges in the graph
- Graph patterns require, allow or disallow particular edges from particular nodes along a path to identify tuning parameters

Example - Statistical Maximum

- *Statistical maximums* are characterized by an expression being assigned to a variable only when the value of that expression would not cause the value of the variable to decrease
- Inference of data transfers and comparisons through method invocation and type hierarchy (Fig. 4) and pattern matching (Fig. 5):

$\mathrm{isActualParameterOf} = \mathrm{isActualParameterOf} \cup (\mathrm{isActualParameterOf} \circ (\mathrm{overrides}^{-1})^{+})$ **(1.1)**

$\mathrm{isAssignedValueOf} = \mathrm{isAssignedValueOf} \cup (\mathrm{isFormalParameterOf} \circ \mathrm{isActualParameterOf}^{-1})$ **(1.2)**

$\mathrm{isAssignedValueOf} = \mathrm{isAssignedValueOf}^{+}$ **(1.3)**

$\mathrm{returns} = (\mathrm{returns} \cup ((\mathrm{overrides}^{-1})^{+} \circ \mathrm{returns}))^{+}$ **(1.4)**

$\mathrm{isComparedGreaterThanOrEqualTo} = \mathrm{isComparedGreaterThanOrEqualTo} \cup (\mathrm{isComparedGreaterThanOrEqualTo} \circ \mathrm{returns})$ **(1.5)**

Figure 4. Inferences using relational algebra create new edges in the graph

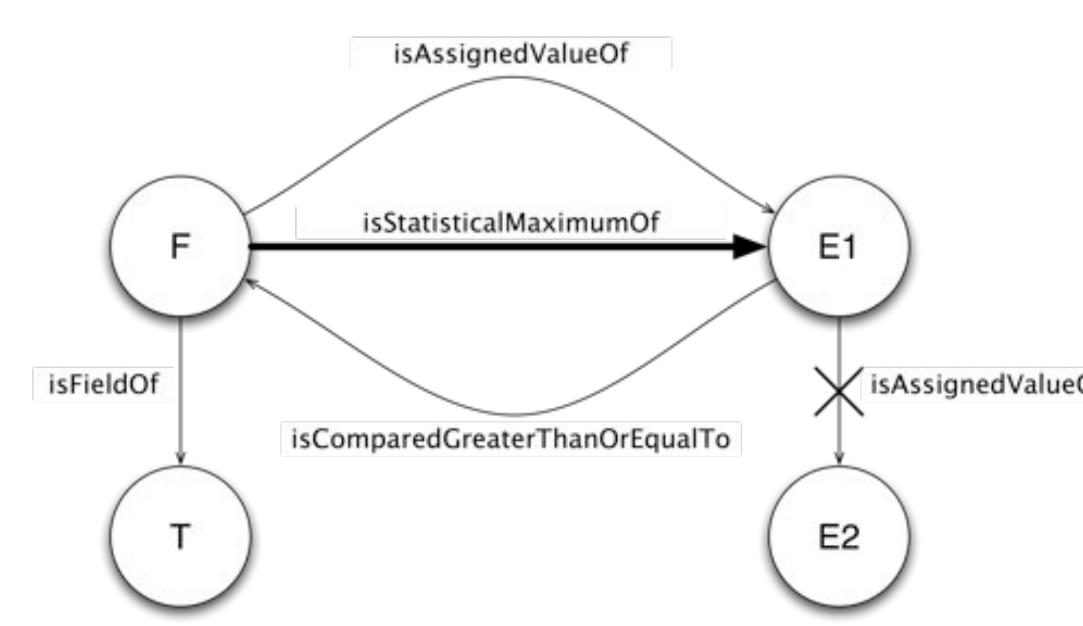

Figure 5. Graph pattern for statistical maximum represented in GraphLog [4]

VI. Conclusions

- Tuning parameters can be automatically identified by matching patterns of use from the taxonomy
- Fact extraction and graph manipulation can be used to get existing relationships from source code and infer new ones
- Patterns identify tuning parameters while also providing clues about related expressions

VII. Future Work

- Integration with management frameworks based on standards such as Web Services Distributed Management (WSDM)
- Refine taxonomy based on expert feedback and study of other application domains
- Orthogonal tuning parameter classification based on resource stereotypes

References

[1] E. Dancy and J.R. Cordy, "STAC: Software Tuning Panels For Autonomic Control," CASCON '06, Toronto, October 2006, pp. 146–160.

[2] A.J. Malton, K.A. Schneider, J.R. Cordy, T.R. Dean, D. Cousineau and J. Reynolds, "Processing Software Source Text in Automated Design Recovery and Transformation," IWPC '01, Toronto, May 2001, pp. 127–134.

[3] R.C. Holt, "Binary Relational Algebra Applied to Software Architecture," Technical Report 345, Computer Science Research Institute, University of Toronto, March 1996.

[4] M.P. Consens and A.O. Mendelzon, "GraphLog: A Visual Formalism for Real Life Recursion," PODS '90, Nashville, April 1990, pp. 404–416.

CASCON 2007, Richmond Hill, Ontario, Canada (October 2007).

This work is supported by an IBM CAS Fellowship and by the Natural Sciences and Engineering Research Council of Canada.