# Software Tuning Panels For Autonomic Control

**Liz Dancy and James R. Cordy**

School of Computing
Queen’s University, Kingston, Ontario, Canada

## I. Introduction

### Autonomic Computing

**Motivation:**

Systems becoming increasingly complex
Approaching a level which may be unmaintainable and unmanageable

**Vision:**

Self-maintenance and tuning in real time
Transparent control on all levels

**Key Ingredients:**

Self-realization, Self-configuration, Self-optimization, Self-Healing, Self protection, Self-Adaptation, Interaction, Hidden complexity

## II. STAC Initiative

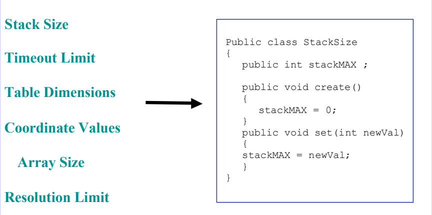
### Background

Tuneable parameters exist in all programs:

- Stack size, limits, table constructs
- Scattered throughout the program for architectural reasons

Selective tuning of such parameters necessary for proper maintenance and control

Need for isolation without loss of function



Stack Size
Timeout Limit
Table Dimensions
Coordinate Values
Array Size
Resolution Limit

```
Public class StackSize
{
   public int stackMAX ;

   public void create()
   {
      stackMAX = 0;
   }
   public void set(int newVal)
   {
   stackMAX = newVal;
   }
}
```

## Goals

- Isolation of tuneable parameters of interest
- Creation of a separate 'control panel' for an entire program
- Provide a framework to automate the rearchitecting of software systems for more efficient autonomic control
- Ability to set, monitor and create tuneable parameters
- Capture concerns which may crosscut several areas in the control panel

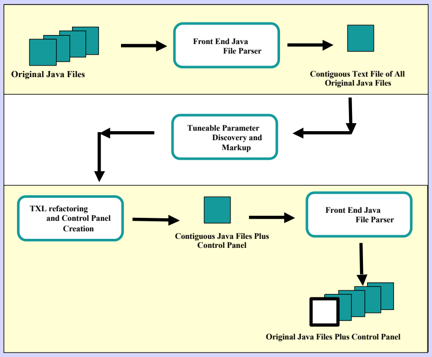
## III. Approach

- Java front end containing hand-coded XML tags denoting tuneable parameters

```
      <control_param>
    int stackSize;
    </control_param>
```

- Program merged into one contiguous source file
- Merged source file transformed using TXL
  - Creates a new Control Panel class containing accessor, mutator and constructor methods for each variable
  - Transforms original code to use references to Control Panel in place of marked up parameters

## Architecture


Original Java Files
Front End Java File Parser
Contiguous Text File of All Original Java Files
Tuneable Parameter Discovery and Markup
TXL refactoring and Control Panel Creation
Contiguous Java Files Plus Control Panel
Front End Java File Parser
Original Java Files Plus Control Panel

## IV. TXL

**Tree Transformation Language**

Hybrid functional and rule-based programming language

Example rule to change a java parameter declaration to a Control Panel reference:

```
rule refCPanelDec  givenType[type_specifier]
                    givenDec [variable_declarator]
   deconstruct givenDec
     Name [variable_name]
    replace [variable_declarator]
     givenDec
    by
     givenType Name = Cpanel.Name.create();
end rule
```

## V. Current Direction

**Implementation Status:**

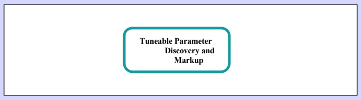
Java and TXL complete
Support for scalar parameters as well as all class references, inheritance referenences and indirect references
Ability to create and modify variables locally in the Control Panel
Plan for simulation and parameter tracking visualization
Plan for attachment of 'origin' variable to each parameter for classification

**Current Limitations:**

Scalar types only
No support for passing references

## VI. Parallel Work

- Identification of parameters of interest across various systems
- Classification of tuneable parameters and their location (mining for interesting parameters)
- Pattern recognition of parameters of interest
- Automated markup to replace manual XML tags
- Classification of tunable parameter behaviour across a program



## VII. Future Work

- Object classification via recursive primitive builds
- Applications for security checks and self-healing
- Extend current setup to apply results of program visualization and data tracking to autonomic self optimization
- Add recursive self-correction of non-sensical values discovered during instrumentation

*Presented at the*
*CSER Meeting Fall 2005*
*Toronto, October 17th*