### Reflectivity Cheat Sheet

#### Defining Reflection...
- **Casually connected.** If the internal structures of a system and the domain they represent are linked so that if one of them changes, the other changes as well. A **reflective system** is then a system which incorporates causally connected structures representing itself.
- **Introspection.** The self-representation of a system can be queried and analyzed.
- **Intercession.** The self-representation of a system can be modified.
- **Reflection = Introspection + Intercession**
- **Meta-objects** describe behavior of base level (i.e., application level) objects, they form a **meta-level**. For example, meta-classes define method lookup.

#### Existing Approaches to Reflection

**Java**
- Structural introspection
- Limited structural intercession, classes not changeable
- Limited behavioral reflection, i.e., objects are wrapped, no interception of method calls or variable access

**Squeak**
- Structural reflection, i.e., classes, methods are objects and dynamically modifiable
- Behavioral reflection, i.e., current execution is reified in thisContext
- **But:** Structural reflection stops at method level!
- **But:** Behavioral reflection limited, reifying execution stack neither efficient nor expressive.

#### Sub-Method Structural Reflection

**Current situation**
- No high-level model for sub-method elements such as message sends, variable accesses
- Different tools use different representations to reason about sub-method elements, but could benefit from a common representation as they heavily communicate with each other.
- Existing representations on the sub-method level are text, bytecode and AST

**Requirements**
- Casual connection high abstraction, extensibility, persistency, efficiency in size and speed.
- **Text:** low level (list of characters), no casual connection
- **Bytecode:** low level (list of integers), not extensible, base level and meta-level code mixed
- **AST:** no casual connection, not extensible, not persistent (generated by compiler, never stored)

**Solution**
- Annotated, persistent AST, bytecode generated on demand
- **Persephone:** Implements reflective methods in Squeak
  - Annotations: either source visible or source invisible.
  - Every node in a method (e.g., message send, variable access, assignment, return statement, ...) can be annotated

#### Partial Behavioral Reflection

**Current situation**
- **Smalltalk:** No model of execution below method body
- **Smalltalk:** Message sending, variable accessing hard-coded in virtual machine
- **MetaClassTalk:** Reflection only controllable at class boundaries
- **MetaClassTalk:** No fine-grained selection (e.g., a specific message send)
- **MetaClassTalk:** Protocol between base and meta level fixed

#### Reflex for Java
- **Hooks:** collection of operation occurrences
- Links: bind hookset to meta-objects, define protocol between base and meta level
- Highly selective reification, flexible meta-level engineering
- **Geppetto:** Reflex in Squeak, based on bytecode transformation (see Figure 1)
- Problems: annotation performance (bytecode mungling), execution transformation (stack manipulation), low-level representation

**Solution**
- Model links as annotations on the AST (see Figure 2)
- Very fast annotations (no decompile)
- On-the-fly code generation
- Generated code is efficient, no stack manipulation

#### Reflectivity in Squeak
- Sub-method structural reflection
- Partial behavioral reflection
- http://scg.unibe.ch/Research/Reflectivity