Java Code Generation

Steve Neuendorffer
UC Berkeley

5th Biennial Ptolemy Miniconference
Berkeley, CA, May 9, 2003

Outline

• Motivation
• Code generation architecture
• Component Specialization
  - Parameter
  - Type
  - Connection
  - Domain
• Token Unboxing and Obfuscation
Design Pressures

Motivation

- System modeling using high-level components enables rapid prototyping
- System implementation becomes the bottleneck
Motivation

- Generation of hardware and software architectures for embedded systems

Ptolemy Classic

CG-VHDL Stars

Stars

CGC Stars

VHDL

Galaxy

(+ scheduling, etc.)

C code
Ptolemy II

Java Actors

Model
(+ scheduling, etc.)

Actor Specializer

JHDL

Java code

C code

Component Specification

Hierarchical Model

Java Code

Finite State Machines

Functional Expressions

Special Purpose Languages

public interface Executable {
  public boolean prefire() throws IllegalActionException;
  public void initialize() throws IllegalActionException;
  public void fire() throws IllegalActionException;
  …
}

actor Switch [T] ()

Integer Select, multi T Input ==> T Output :
  action Select: [i]. Input: [a] at i ==> [a] end
  end

Expression2

signal * carrier + noise

actor Switch [T] ()

Integer Select, multi T Input ==> T Output :
  action Select: [i]. Input: [a] at i ==> [a] end
  end
Parameter Specialization

Here the scale factor has not been determined yet, because it depends on the parameter x.

Implicit vs. Explicit information

Explicit Specialization

Implicit Specialization

Neuendorffer/UC Berkeley/Ptolemy Miniconference 9
Parameter Specialization

Implicit Parameter Specialization relies on model analysis to determine parameter values that set and cannot change.

Dynamic parameters:
- Parameters accessible through a user interface.
- Parameters that can be set in the FSM transitions.
- Parameters with values depending on unbound variables

All other parameters can be specialized using implicit context.

Type Specialization

Implicit analysis simply uses the standard type inference mechanism.

Currently assume that even when parameter values change, types do not.
**Aggregation**

Parameter and Type specialization can be performed on individual actors.

Domain and Connection specialization occur as part of aggregation.

---

**Connection Specialization**

Connection specialization ties actors directly to the channels they are connected to.

Connections are assumed not to change.
Connection Specialization

Connection specialization ties actors directly to the channels they are connected to.

Domains are assumed not to change.

Domain Specialization

Connection specialization ties actors directly to the channels they are connected to.

Connections are assumed not to change.
Domain Specialization

Connection specialization ties actors directly to the channels they are connected to.

Domains are assumed not to change.

Token Unboxing

- After specialization, memory use is a significant performance bottleneck.
- Token Unboxing removes allocation of token objects by replacing each token with its constituent fields.
Obfuscation

Java .class files contain a large number of strings
  • String literals
  • Class names
  • Method signatures
  • Field signatures
  • Exception messages

Obfuscation renames these strings to shorter ones, where possible.

Reduces bytecode side.

Why does this all work?

  • Ptolemy actor specifications are highly polymorphic and reusable.

  • However, we commonly use them only in monomorphic contexts.
    - Constant, exactly analyzable types.
    - Connections, domains don’t change.
    - Parameter values change only in known patterns.
Why does this all work?

- We’ve eliminated a large amount of *synchronization* overhead.
  - Workspace.getReadAccess()
  - Workspace.doneReading()
- We’ve eliminated *object allocation*, which reduces load on the garbage collector.
- Generated code is entirely *self contained*. Functionality is important, interfaces are not.

Capabilities

- Applications
  - Control algorithm for Caltech vehicles.
  - Rijndael encryption algorithm.
  - HTVQ Video compression.
- Supported
  - Expression actor
  - FSM actor
  - Modal models
  - SDF and Giotto domains
- Not supported
  - Record types
  - Transparent hierarchy
How to use

Command-line interface
   >> copernicus model.xml

Code is generated in:
   $PTII/ptolemy/copernicus/java/cg/model/

Vergil User interface
   view -> Code Generator
   Allows easier changing of parameters.

Conclusion

Java code generation is at the point where it might be useful for speeding up the simulation of some models.

Current work:
   Embedded Java platform
   Integration with hardware synthesis
   Guided refinement