

Code Generation in Ptolemy II

Jeffrey Tsay, Shuvra S. Bhattacharyya¹,
Christopher Hylands², and Edward A. Lee²

¹ University of Maryland at College Park

² University of California at Berkeley

4-th Biennial Ptolemy Miniconference
March 22-23, 2001, Berkeley, CA

Motivation

- **Limitations of Ptolemy Classic code generation**
 - Separate domains for code generation
 - Separate actor source code
 - Inefficient inter-actor communication
- **Analysis of component internals for improved compilation, and translation to multiple target languages**
 - Explored in El Greco (Buck/Vaidyanathan)
- **Reduce run-time overhead of type-polymorphic actors**
- **'Shallow' code generation mode to construct Ptolemy-dependent Java class definitions of Ptolemy models**
 - Automate the construction of applets from Ptolemy models

Deep Code Generation Example

Original actor source code

```
Token t1 = in.get(0);  
Token t2 = in.get(1);  
out.send(0, t1.multiply(t2));
```

Specialize token decs.

Actor code with specialized tokens

```
IntMatrixToken t1 = in.get(0);  
IntMatrixToken t2 = in.get(1);  
out.send(0, t1.multiply(t2));
```

Transform Ptolemy semantics

```
int[][] t1 = _cg_in_buf[0]  
[_offset = (_offset + 1) % 5];  
// ... similarly for t2  
_cg_out_buf[_out_offset =  
(_out_offset + 1) % 8] =  
IntegerMatrixMath.multiply(t1, t2);
```

```
int[][] t1 = _cg_in_buf[0]  
[_offset = (_offset + 1) % 5];  
// ... similarly for t2  
_cg_out_buf[_out_offset =  
(_out_offset + 1) % 8] =  
t1 * t2;
```

Code Generation in Ptolemy II, 3