The Caltrop Actor Language

Johan Eker
UC Berkeley

MoBIES group, Carnegie Mellon, November 30, 2001
What is an Actor?

- Input & output ports
- States
- Parameters
- Consumes and produces tokens
- An actor is said to be fired
Why Another Language?

• Java is current practice
  – No need to learn another language
  – The functional behavior of the actor is mixed with platform issues
  – Most of the actor code deals with administration

• Caltrop offers a concise, more abstract description
  – Simplifies code generation
  – Allows some analysis
  – Quick to implement actors, “throw away actors”
  – Protects IP - Insulates the actor from the platform
Caltrop Actor Language

• Imperative, but with a functional flavor
• Embedded in a host language
  – All data types are imported
• Operates on token streams

\[
\text{actor Add () double A, double B } \rightarrow \text{ double C:} \\
\text{action } [a], [b] \rightarrow [c]: \\
\quad c := a + b; \\
\text{end} \\
\text{end}
\]

A: [1, 2, 3, ...], B: [1, 2, 3, ...] \Rightarrow C: [2, 4, 6, ...]
Type Polymorphism

- Type parameters

```actor
Add [T1, T2]() T1 A, T1 B → T2 C:
where T2 > T1:
action [a], [b] → [a+b]: end
end```

- Type lattice not part of the language
- Operators are injected from the environment
- Type checking and/or type inference
Actions and Patterns

- Matches patterns on input ports

    actor MyActor () double A, double B ➔ double C:
    action [a, b], [c] ➔ [a+b+c]: end
    end

    - Assume the following sequence on both ports
      [1,2,3,4,5,.....], the output would then be the
      sequence [1+2+1,3+4+2,...]=[4,9,14,...]

    - The pattern [a, b | c] binds a and b to the two
      first tokens, and c to the rest of the input
      sequence
Actions and Patterns cont’d

• One actor, many actions

```plaintext
actor Sum () double A, double B ➔ double C:
  action [a], [b] ➔ [a+b]: end
  action [a], [] ➔ [a]: end
  action [], [b] ➔ [b]: end
end
```

• Conditional actions

```plaintext
actor Abs () double A ➔ double B:
  action [a] ➔ [a] where a > 0: end
  action [a] ➔ [-a] where a <=0: end
end
```
Multi ports

- A port may have several channel
  - A multi ports is a map from \textit{ChannelID} to data type
  - Syntax: \textless pattern\textgreater \textless expression\textgreater

- The Switch actor
  - One selector input port
  - One data multi input port

\begin{verbatim}
actor Switch [T] () int i, multi T input ➔ T out:
  action [i], [a] i ➔ [a]: end
end
\end{verbatim}
actor Add [T] () multi T input ➔ T out:
  action [a] any ➔ [sum] with T sum := 0:
    foreach int i in dom a :
      sum := sum + a[i];
    end
  end
action [a] {1,2,3} ➔ [sum] with T sum := 0:
  foreach int i in dom a :
    sum := sum + a[i];
  end
end
end
Behavior polymorphism

- The semantics for Caltrop is not complete
- Different interpretation for different targets, i.e. the same Caltrop block will have different meaning in Simulink and Ptolemy II/SDF. For example:

```plaintext
actor MyActor () int in1, int in2 \rightarrow int out:
  action [a], [b] \rightarrow [a+b]: end
  action [a], [] \rightarrow [2*a]: end
  action [], [b] \rightarrow [2*b]: end
end
```
Analysis

- Production and consumption rates may be extracted by inspection of patterns

```plaintext
actor MyActor () int in1, int in2 → int out1, int out2:
  a1: action [a], [b] → [a], [b]:
      end
  a2: action [a], [] → [a, a], []:
      end
  a3: action [], [b] → [], [b, b]:
      end
end
```

![Diagram of actor interactions](image)
Composing Caltrop Actors
Behavioral Restrictions

- Restrict the allowed actions
- Regular expressions over actions

```latex
\begin{verbatim}
actor MyActor () int in1, int in2 \rightarrow int out1, int out2:
  a1: action [a], [b] \rightarrow [a], [b]: end
  a2: action [a], [] \rightarrow [a, a], []: end
  a3: action [], [b] \rightarrow [], [b, b]: end

selector
  a1 (a2|a3)*
end
end
```

- The first action must be \texttt{a1} which then is followed by either \texttt{a2} or \texttt{a3} zero or more times
The Rest of Caltrop

- Expressions are side effect free
- Lambda closures
- Statements
  - foreach
  - while
  - if-then-else
- Built-in types: list, map, set
Current Status

- 2 Post-docs + 2 graduate students
- Compiler and interpreter in the works
- http://www.gigascale.org/caltrop