A Coordination Model and Algebra for Domain Polymorphism
preliminary results

Farhad Mavaddat
ERL/EECS
UC Berkeley\textsuperscript{a}

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

\textsuperscript{a}On leave from University of Waterloo, Canada.

---

Goal and application issues:
goals:
- unified model of polymorphic components,
- formalizing coordination domains,
- reducing coordination overhead,
  - light-weight co-ordination,
  - compile-time vs execution time issues.

application domains:
- embedded systems
- signal processing
- co-design (hwr and swr)
Time model:

Port types:
Given port $P$:

- with reference to an output port:
  - $\tilde{p}$ a continuously (time) varying source.
  - $\bar{p}$ a latched source.
  - $p$ a hand-shaking source.

- with reference to an input port:
  - $\tilde{p}$ sensitive to continuous input excitations.
  - $\bar{p}$ sensitive to input at clock event,
  - $p$ a hand-shaking sink.
Dependencies:

- **Internal:** between the input “a” (or  \( \dddot{a} \) or  \( \dddot{a} \)) and output “b” (or  \( \dddot{b} \) or  \( \dddot{b} \)) of a module.
  - *isochronic:* written as “\( a/b \)”, \( a \) is available for generation of \( b \) in a predictable way.
  - *sequential:* written as “\( a, b \)”, delayed influence of \( a \) on \( b \), by an unknown number of clock *ticks,*

- **External:** between environment and input \( a \). Written as \( a/— \) (or \( a/? \)), indicates \( a \)’s sensitivity to the environment. Timing depends on the type of \( a \).

- **Interface:** between the output \( b \) and the environment.
  Written as \( —/b \) (or \( b! \)), indicates a source ready to influence the environment. Timing depends on the type of \( b \).

**Intra module:** between the input “\( a \)” (\( \dddot{a} \) or \( \dddot{a} \)) and output “\( b \)” (\( \dddot{b} \) or \( \dddot{b} \)) of a module:

\( \dddot{a}/\dddot{b} \): e.g. an ideal adder,
\( \dddot{a}/\dddot{b} \): e.g. a sample-and-hold (S&H) device
\( \dddot{a}/\dddot{b} \): e.g. output of an integrator
\( \dddot{a}/\dddot{b} \): e.g. a shift register,
\( \dddot{a}/— \): also written as \( \dddot{a}? \), continuous software input
\( \dddot{a}/— \): also written as \( \dddot{a}? \), clocked input to software,
\( \dddot{a}/— \): also written as \( \dddot{a}? \), a hand-shaked input.
\( —/\dddot{b} \): also written as \( \dddot{b}! \), continuous software generated output
\( —/\dddot{b} \): also written as \( \dddot{b}! \), a software generated register output,
\( —/\dddot{b} \): also written as \( \dddot{b}! \), a hand-shaked output,
Module examples:

```
\begin{figure}
\centering
\begin{tikzpicture}
\node (A) at (0,0) [fill,circle,inner sep=2pt]{};
\node (B) at (2,0) [fill,circle,inner sep=2pt]{};
\draw (A) edge [bend left=45] node [midway, above] {$\tilde{x}/\tilde{z}$} (B);
\draw (B) edge [bend right=45] node [midway, above] {$\tilde{y}/\tilde{z}$} (A);
\end{tikzpicture}
\caption{Diagram 1}
\end{figure}
```

synchronous reactive example:

```
\begin{figure}
\centering
\begin{tikzpicture}
\node (A) at (0,0) [fill,circle,inner sep=2pt]{};
\node (B) at (2,0) [fill,circle,inner sep=2pt]{};
\draw (A) edge [bend left=45] node [midway, above] {$\tilde{c}/\tilde{d}$} (B);
\draw (B) edge [bend right=45] node [midway, above] {$\tilde{d}/\tilde{e}$} (A);
\end{tikzpicture}
\caption{Diagram 2}
\end{figure}
```
**Inter module:** Co-ordination:

- $\ddot{a}/\ddot{a}$: current source value passed to sink (wire),
- $\ddot{a}/\ddot{a}$: source value at clock tick passed to sink,
- $\ddot{a}/\dot{a}$: source value at (one sided) hand-shake passed to sink,
- $\ddot{a}/\ddot{a}$: *constant* source value passed to sink,
- $\ddot{a}/\dot{a}$: source value at clock tick passed to sink,
- $\ddot{a}/\ddot{a}$: latest source passed to (one sided) hand-shaking sink,
- $\dot{a}/\dddot{a}$: source value at (one sided) hand-shake passed to sink?,
- $\dot{a}/\dddot{a}$: source value at (one sided) hand-shake passed to sink?,
- $\dddot{a}/\dddot{a}$: value passed at the time of (two-sided) hand-shake.

---

**Coordination Basics:**

- “†”: The “blocking” operator has two types: *internal* and *external*.
  - An *internal* block is generated by a running application process and shown by an edge leaving a computation state.
  - An *external* block indicates waiting for an external (blocking) event and shown by an edge leaving a blocked state.
- “\(\downarrow\)”: The “unblocking” operator has two types: *internal* and *external*.
  - An *internal* unblock is generated a running coordination process. This is shown by an edge leaving a computation state.
  - An *external* block is waiting for an external (unblocking) event. This is shown by an edge leaving a blocked state.
coordination basics:

\[ C \xrightarrow{f} B \]

\[ C_a \xrightarrow{f} C_b \]

\[ P \xrightarrow{e} Q \]

4-th Biennial Ptolemy Miniconference
a/a abstraction:

SDF:
PN:

```
<table>
<thead>
<tr>
<th>p</th>
<th>e</th>
<th>Q</th>
</tr>
</thead>
</table>
```

```
| ot | in |
```

Farhad Mavaddat

ERL of EECS of UCB

4-th Biennial Ptolemy Miniconference

Design example: software+synchronous components:

```
\[ \sigma d/\varepsilon/\tilde{e}/\tilde{a}/[\tilde{a}/\varepsilon]/\tilde{b}/\tilde{b}/\varepsilon/\tilde{e}/\tilde{d}/\tilde{d}/\tilde{e}, \tilde{a}/\tilde{d}/\tilde{d}/\tilde{e} ] \]
```

Farhad Mavaddat

ERL of EECS of UCB