Modelling Timed Concurrent Systems Using Activity Diagram Patterns

Étienne André†, Christine Choppy†, Thierry Noulamo‡

†Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, France
‡University of Dschang, IUT Fotso Victor, LAIA, Bandjoun, Cameroun
Introduction

- UML activity diagrams: rich, permissive syntax, favours mistakes
- Semantics in natural language prevents formal verification
Introduction

- UML activity diagrams: rich, permissive syntax, favours mistakes
- Semantics in natural language prevents formal verification

Our contributions [A., Choppy, Noulamo (KSE’14)]

- Activity diagram patterns:
  - TADC (Timed Activity Diagram Components)
- Modular composition mechanism (also refinement)
- Semantics with time Petri nets
Outline

1. Timed Activity Diagrams Patterns
2. Translation into Time Petri Nets
3. Conclusion and Perspectives
Outline

1. Timed Activity Diagrams Patterns
2. Translation into Time Petri Nets
3. Conclusion and Perspectives
UML Activity Diagrams

- initial node, activity/flow final nodes, decision nodes (guards), fork/join nodes
- Global variables typed with **finite domains** (e.g. enumerated types)
- Activities may involve global variables **discrete, instantaneous modifications** (assignment, function call with side-effects)
Timed Activity Diagram Components (TADC)

- input connectors, output connectors
- “well-formed” activity diagrams, restricted construct set, adding timed constructs
- modular specification with possible refinement
- inductive mechanism
10 TADC patterns

- initial node, flow final node, simple activity ...
10 TADC patterns

- initial node, flow final node, simple activity ...
- sequence, non-deterministic delay, deterministic delay, deadline,

sequence

input connectors = TADC1 input connectors
output connectors = TADC2 output connectors
10 TADC patterns

- initial node, flow final node, simple activity ...
- sequence, non-deterministic delay, deterministic delay, deadline,

sequence, deterministic delay $d \in \mathbb{R}_{\geq 0}$, non-deterministic delay $[0, d]$, deadline
10 TADC patterns

- initial node, flow final node, simple activity...
- sequence, non-deterministic delay, deterministic delay, deadline,

TADC_1 execution
- activity A
- TADC_2 starts after at most d units of time
- alternatively, TADC_3 starts after exactly d units of time

deadline
10 TADC patterns (followed)

- decision, merge, synchronisation (cf UML)
Outline

1. Timed Activity Diagrams Patterns
2. Translation into Time Petri Nets
3. Conclusion and Perspectives
Time Petri Nets [Merlin, 1974] with global variables

- A kind of automaton
  - Bipartite graph with places ("standard" tokens) and transitions
  - Transitions associated with a firing interval
  - Enabled (tokens present) transitions must fire at the end of their firing interval (unless meanwhile disabled by another transition)
  - Global variables typed by a finite domain, used to express guards, updated when transition fired
  - Global variables: syntactic extension only
  - Note: Time Petri Nets with global variable: similarities with coloured Petri nets

```
p1

1, 3 {¬ b} 2, 4 i := i + i

p2
p3
```
Time Petri Nets [Merlin, 1974] with global variables

- A kind of automaton
  - Bipartite graph with places ("standard" tokens) and transitions
  - Transitions associated with a firing interval
  - Enabled (tokens present) transitions must fire at the end of their firing interval (unless meanwhile disabled by another transition)
  - Global variables typed by a finite domain, used to express guards, updated when transition fired
  - Global variables: syntactic extension only
  - Note: Time Petri Nets with global variable: similarities with coloured Petri nets

\[
\begin{align*}
0 & \text{[1,3]} \quad \{\neg b\} \\
1 & \text{[2,4]} \\
2 & \text{i := i + i}
\end{align*}
\]
Translation mechanism

- each TADC is translated into a TPN fragment where the connectors are translated into places
- two TPN fragments can be composed by fusing the corresponding connector places together
- simple activity: TPN transition connected to input and output places (to be used for composition)

![Diagram of a simple activity]

- assignments are easily translated
- functions involving a user input $\rightarrow$ non-deterministic choice
Translation mechanism (cont’d)

- Deterministic delay

\[
\text{Tr}(\text{TADC}_1) \xrightarrow{[d, d]} \text{Tr}(\text{TADC}_2)
\]
Translation mechanism (cont’d)

- Deadline

\[ T_{ADC1} \xrightarrow{A} T_{ADC2} \xrightarrow{d} T_{ADC3} \xrightarrow{} Tr(T_{ADC1}) \xrightarrow{} [0, d] \xrightarrow{} [d, d] \xrightarrow{} Tr(T_{ADC2}) \xrightarrow{} Tr(T_{ADC3}) \]
Translation mechanism (cont’d)

- **Decision**

\[
\text{Tr}(\text{TADC}) \rightarrow \text{Tr}(\text{TADC}_1) \cdots \text{Tr}(\text{TADC}_n)
\]
Outline

1. Timed Activity Diagrams Patterns
2. Translation into Time Petri Nets
3. Conclusion and Perspectives
Conclusion

- Activity diagram patterns with **timed** constructs for **modular** composition.
- **Semantics** in terms of time Petri nets.
- [A., Choppy, Reggio (SER'A’13)]: “precise” activity diagram patterns to model business processes, modular, coloured Petri nets semantics.
- Here: focus on time extension, less restrictive patterns (arbitrary number of input and output connectors), easier to “plug in” a higher level scheme.
## Conclusion

<table>
<thead>
<tr>
<th>Element</th>
<th>[UML 2.5]</th>
<th>[ACR13]</th>
<th>This work</th>
<th>[ACN14]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>Limited</td>
<td>Yes</td>
<td>Limited</td>
<td></td>
</tr>
<tr>
<td>Participants</td>
<td>Limited</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Initial / final nodes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Decision</td>
<td>Yes</td>
<td>Restricted</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Merge</td>
<td>Yes</td>
<td>Restricted</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Fork</td>
<td>Yes</td>
<td>Restricted</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Join</td>
<td>Yes</td>
<td>Restricted</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Timed transitions</td>
<td>Limited</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Summary of the syntactic aspects considered
Perspectives

- Enrich with more complex features, e.g., timed synchronization of activities
- Refinement
- Tool ...
Bibliography
Modelling timed concurrent systems using activity diagram patterns.
In Nguyen, V., Le, A., and Huynh, V., editors, Proceedings of the 6th International
Conference on Knowledge and Systems Engineering (KSE’14), volume 326 of

Activity diagrams patterns for modeling business processes.
In SERA, volume 496 of Studies in Computational Intelligence, pages 197–213.
Springer.

A study of the recoverability of computing systems.
PhD thesis, University of California, Irvine, CA, USA.

OMG unified modeling language. version 2.5 beta 2, 2013-09-05.