Component-Based Behavioural Modelling with High-Level Petri Nets

Rémi Bastide, Eric Barboni
LIIHS – IRIT, University of Toulouse, France
{bastide, barboni}@irit.fr
Software Components

- Active domain for industry, commercial success
- Several mature component models competing: .Net, JavaBeans, CCM
- Common concepts:
  - Component as a black box, accessed through software interfaces
    - Favor composition over inheritance (white box reuse)
  - Multicast, event-based communication as well as unicast method invocation
  - Design-time assembly and configuration of components to produce an executable system
- Driven by industry, lack of formal foundations
Our research goals

• Define a component model
  • Not specially original, close to CCM
• Propose a formal notation to specify the inner behavior of components
  • Petri nets (of course !)
  • Allows for specifying concurrent, event and time-driven behavior
• Define a mapping from the constructs of the component model to the Petri net
• Provide a formal definition of inter-components communication
  • Method invocation and event multicast
  • Petri nets (again !)
• Provide a denotational semantics for an assembly of components
  • From their internal behavior and inter-communications
  • Provided as an unstructured high-level Petri net
Outline of the talk

• **Introduction**
  • Done

• **The component model**

• **Mapping to Petri nets**

• **A case study**
The Component Model

- The *Envelope* of a component is made of a set of *Ports*
- Each port described by a pair (Name, java interface)
  - Facets
    - Functional features offered
    - May be connected to several receptacles
  - Receptacles
    - Functional features required
    - Requires exactly one facet
  - Event sources*
    - Events the component can emit
    - Each method in the event interface represents an event
    - Connected to any number of event sinks
  - Event sinks*
    - Events the component is willing to receive
    - Connected to any number of event sources
- * Some syntactic constraints on interfaces
Example: a 2D gesture recognizer

```java
public interface MouseListener extends EventListener {
    public void mouseClicked(MouseEvent e);
    public void mousePressed(MouseEvent e);
    public void mouseReleased(MouseEvent e);
    public void mouseEntered(MouseEvent e);
    public void mouseExited(MouseEvent e);
}
```

```java
public interface MouseMotionListener extends EventListener {
    public void mouseDragged(MouseEvent e);
    public void mouseMoved(MouseEvent e);
}
```

```java
public interface MouseInputListener extends MouseListener, MouseMotionListener {
}
```

```java
public interface ActionListener extends EventListener {
    public void actionPerformed(ActionEvent e);
}
```

```java
public interface GestureRecognizer {
    public void start();
    public void stop();
}
```
Assemblies: wiring components together
Mapping for Facets and Receptacles

```java
public interface Buffer {
    void put(String m);
    String get();
}
```

Diagram:
- **Consumer** connected to **myBuffer** connected to **Producer**
Specifying Server-side behaviour (myBuffer)

```java
public interface Buffer {
    void put(String m);
    String get();
}
```

buffer: Buffer

myBuffer
Specifying client-side behaviour (e.g. Consumer)
Merging Facets and Receptacles
public interface ActionListener
extends EventListener {
    void actionPerformed(ActionEvent e);
}
Specifying sender-side behaviour (event source)

```java
public interface ActionListener extends EventListener {
    void actionPerformed(ActionEvent e);
}
```

Sender

action: ActionListener
Expected semantics

- The event emitter can always emit an event through its event sources whether or not there are connected receivers.
- Event emission is non-blocking and atomic (for the emitter).
- Transmission of events from source to sink may take time.
Specifying receiver-side behavior (event sink)

Several synchronized transitions (Moalla et al. 78) may be associated to each event of a sink.

- Synchronized transitions fire (if enabled) only when they receive an external signal.
- If a signal occurs when no transition is enabled, it is lost.

```java
class ActionListener extends EventListener {
  void actionPerformed(ActionEvent e);
}
```

Diagram showing transitions and actions associated with events.
Merging event sources and sinks

- **Using signal arcs** from Signal nets (Starke & Roch, 2001)
- Several good analysis techniques developed for this class of nets

![Diagram showing signal arcs connecting event sources and sinks]
Case study: Head gesture recognition

Typical head nod (above) and shake sequence (bottom) of varying duration, intensity, energy and energy rates.
Assemblies and Hierarchy
Petri net-based behavioral specification
Conclusion

• A component model close to the practice of Software Engineering, yet provided with a simple formal semantics in terms of HLPN

• Connectors (Facet-Receptacle, Source-Sink) can be considered as PN composition operators

• Partially implemented
  • Facet-Receptacle close enough to our previous work on CORBA
  • Need a proper assembly editor

• Provide an environment where the behavior of a component may be seamlessly provided in Java, in PN, or as an assembly of sub-components
  • Perform formal verification on parts of the system that particularly deserve it