Eclipse Coordination Tools

Software Development at SEN3

Christian Koehler

Centrum voor Wiskunde en Informatica (CWI)
Amsterdam, The Netherlands

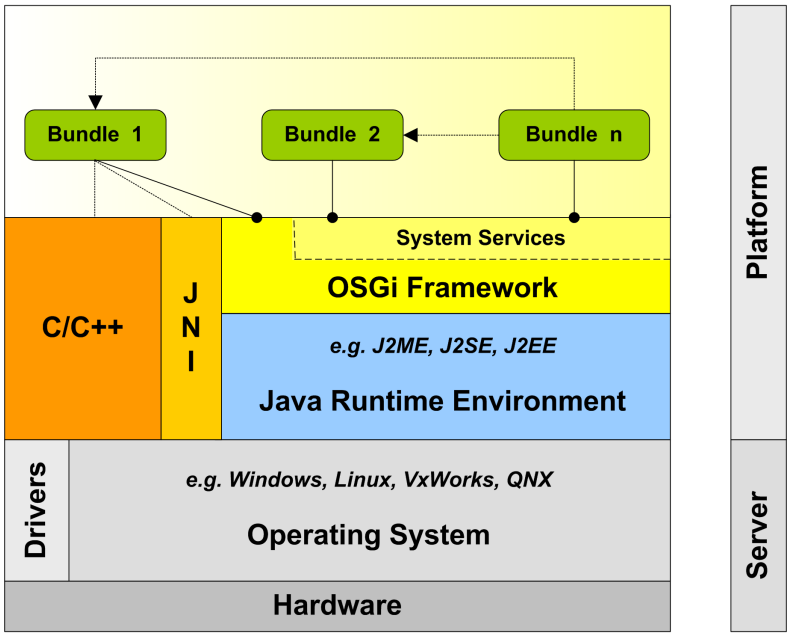
17th October 2007

Outline

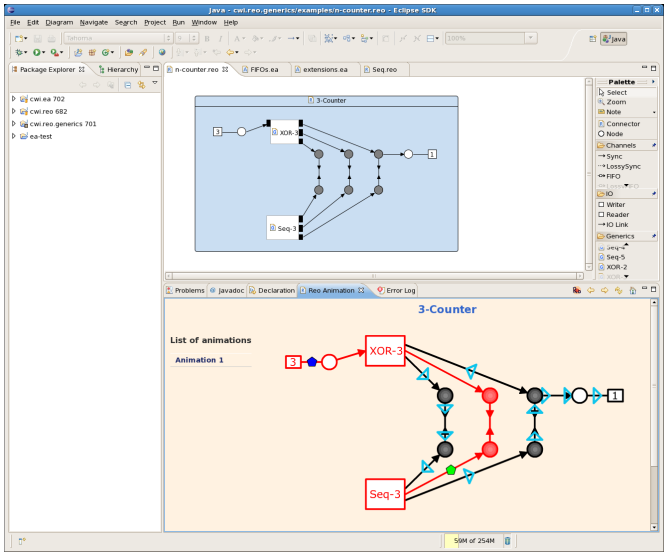
Outline

Eclipse Platform

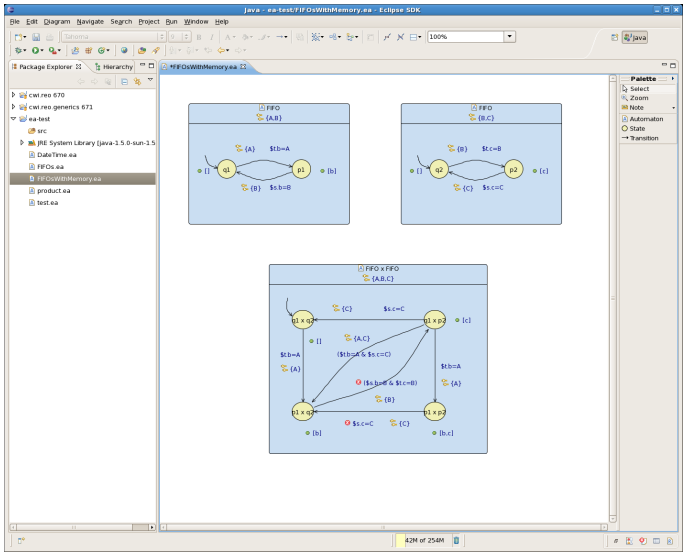
- Open-source, platform-independent software framework.
- Sophisticated plug-in architecture.
- Meta-tools for development of graphical and textual editors.
- Integration of heterogeneous tools possible.



Eclipse Coordination Tools (ECT)

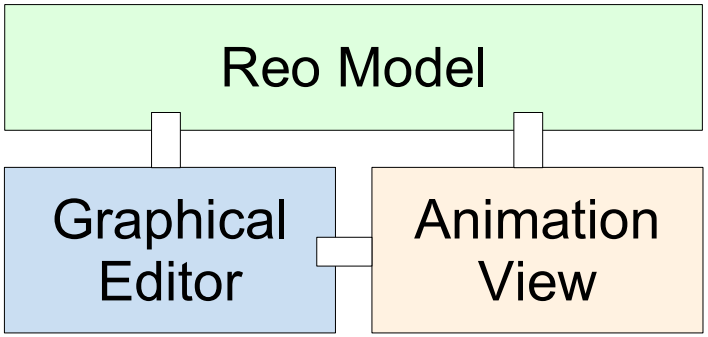


Reo Tools



Automata Tools

Reo Tools



- Edit connectors graphically and save them as XML/XMI.
- Generate Flash animations of connectors on the fly.
- Extend the Reo model to define your own channel types.
- Model check Reo circuits and constraint automata.

Reo Editor

Supported channel types:

- Sync, LossySync
- FIFO, LossyFIFO
- SyncDrain, AsyncDrain
- SyncSpout, AsyncSpout

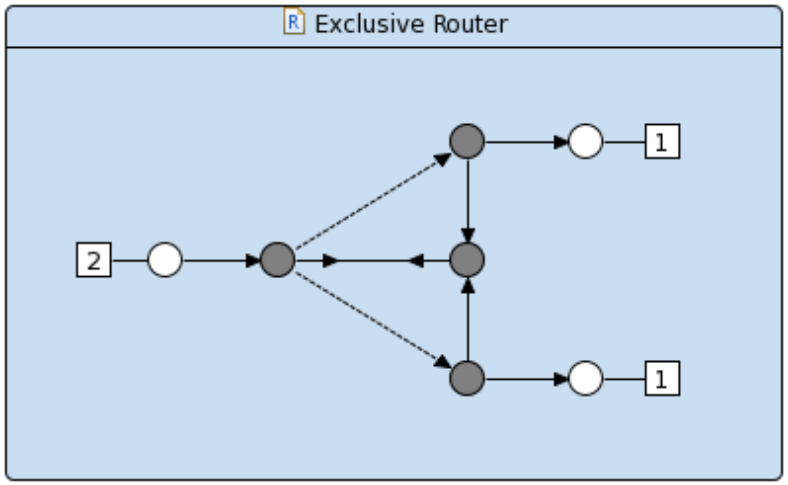


Figure: Exclusive router.

Reo Editor

Other primitive connectors:

- Writer, Reader
- Replicator, Merger
- ...

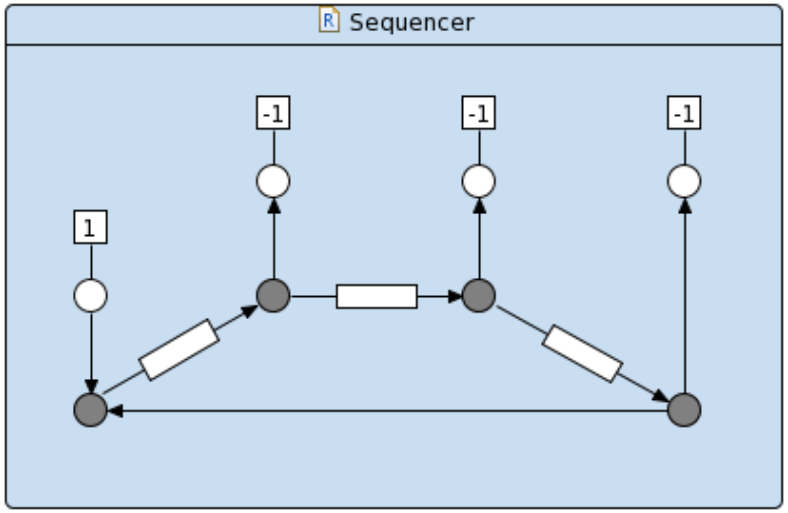


Figure: Sequencer.

Generic Primitives

Support for Generics:

- Connectors can be exported to a library.
- Internal structure and behaviour is hidden.
- Reuse connectors to build more complex ones.

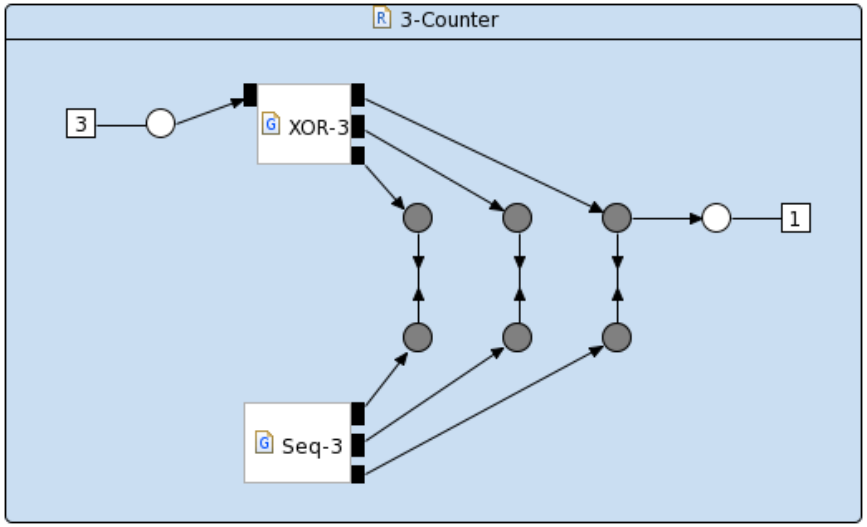


Figure: 3-Counter circuit.

$\Rightarrow$ Modularization and encapsulation.

Reo Animation

Compositionally computed animations :

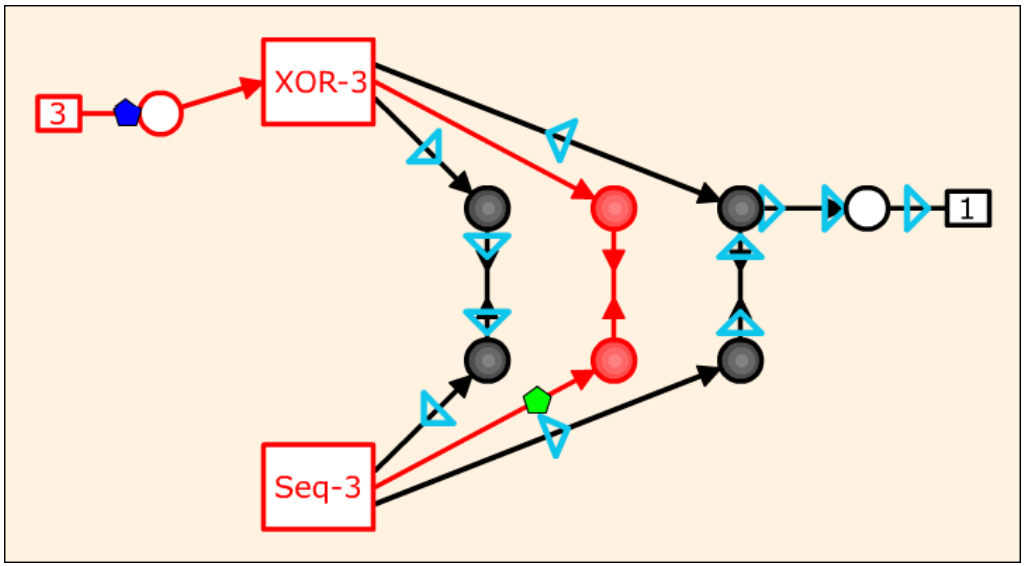


Figure: Animated 3-Counter.

Integrated Model Checker

Model Checking:

- Efficient symbolic model checking implemented in C++.
- Joint project with TU-Dresden.

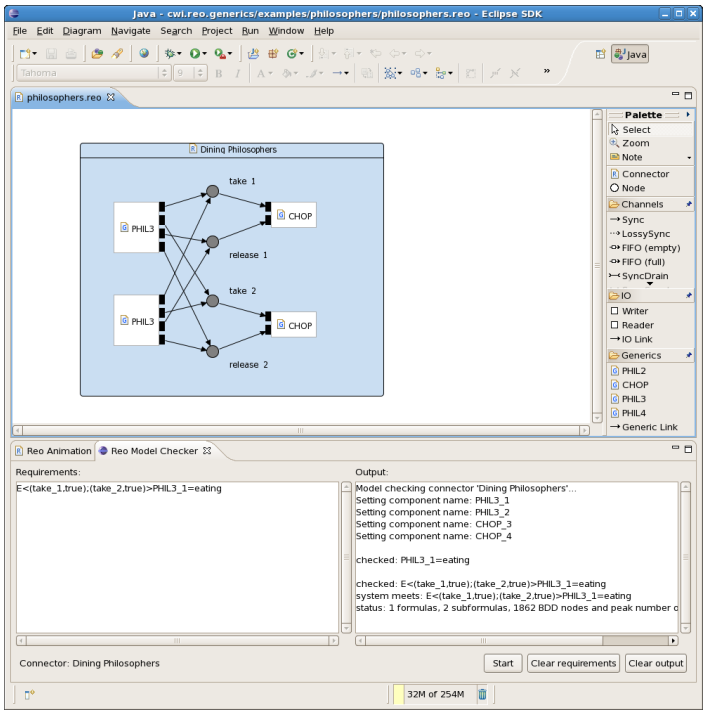


Figure: Dining philosophers.

Dynamic Reo

Need for dynamic coordination:

- Software systems evolve over time.
- Static connectors are not sufficient for complex applications.
- New paradigms demand highly dynamic adaptations (e.g. in SOA or Grid computing).
- Deployment and reconfiguration procedures are essential for a reliable coordination environment.

High-Level Transformations

Connector Reconfiguration:

- Support for local and global changes.
- Pattern-based transformation rules.
- Perform complex refactorings in an atomic step.

$\Rightarrow$ Need for a high-level transformation language.

SOA Example

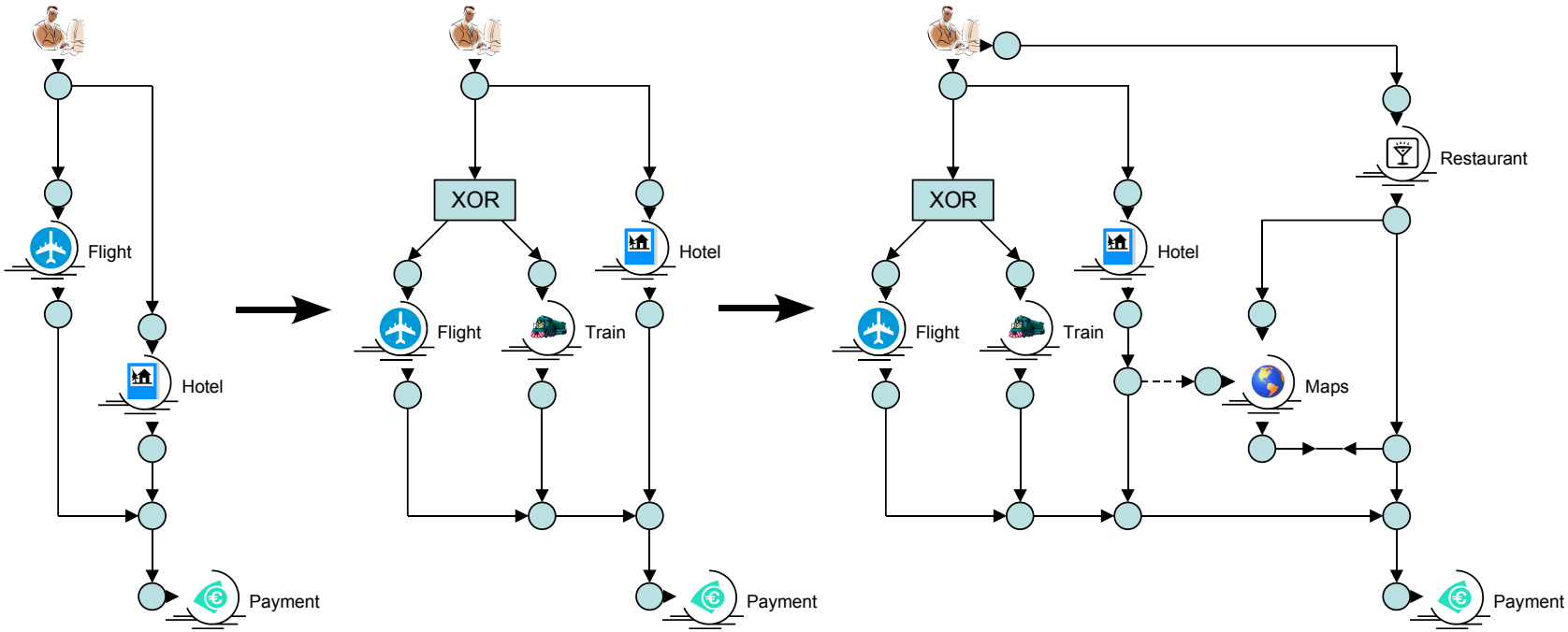


Figure: SOA process customization.

Graph Transformation

- Modeling structure or behavior is often done with some sort of graphs.
- Model transformation can be realized as a transformation of graphs.
- A set of transformation rules constitutes a graph grammar.

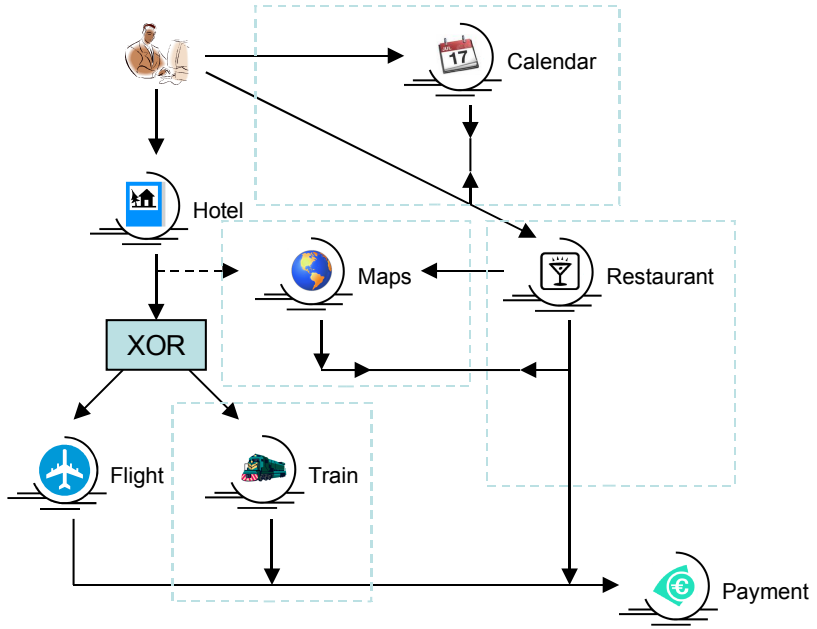


Figure: SOA process.

Process Customization

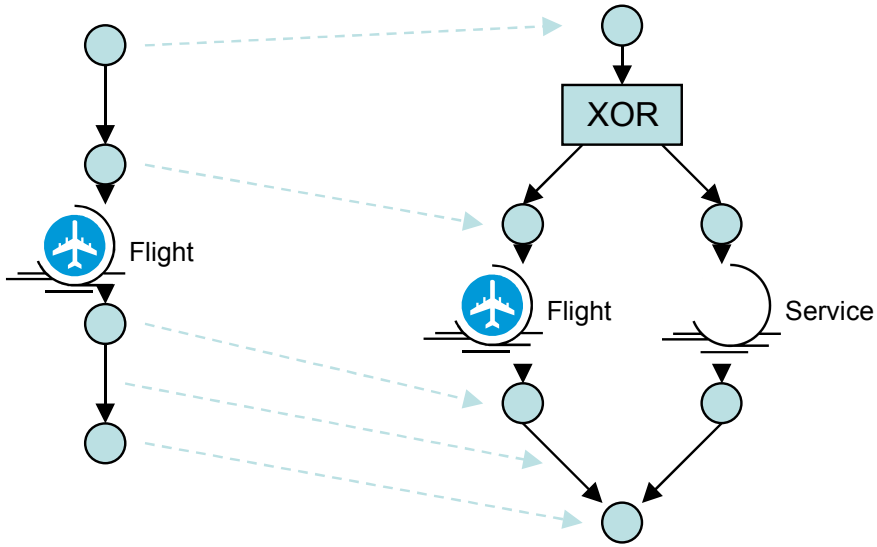


Figure: Rewrite rule *"Add flight alternative"*.

EMF Model Transformation (EMT)

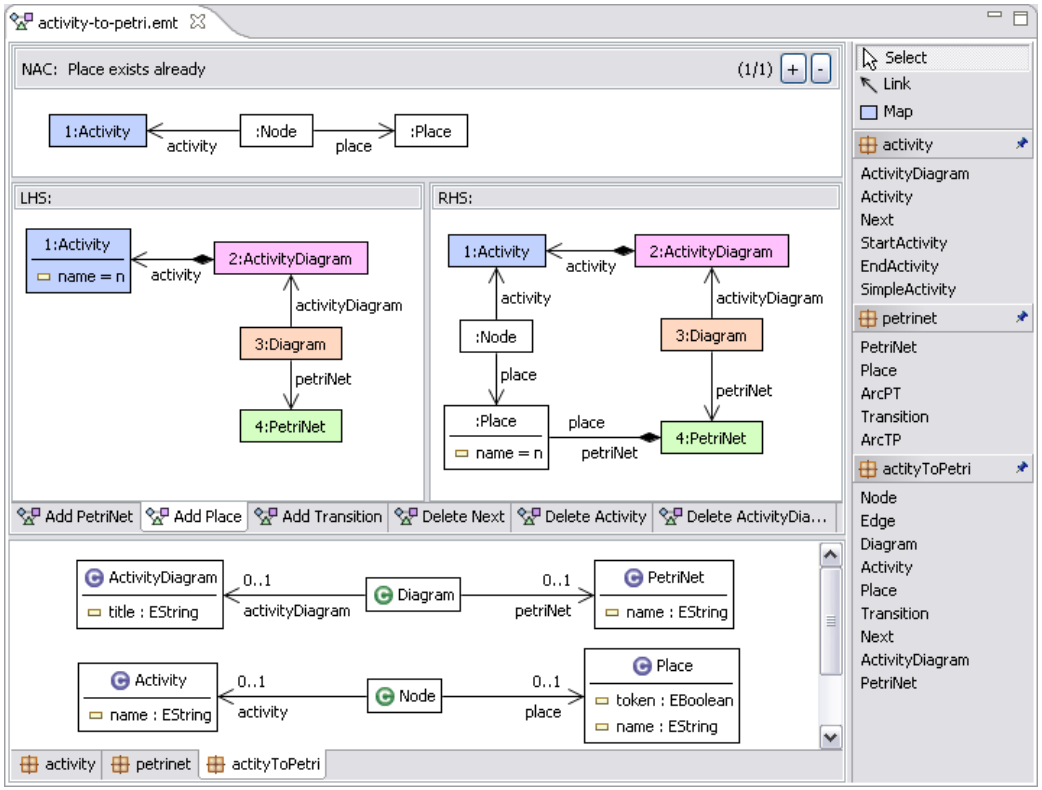
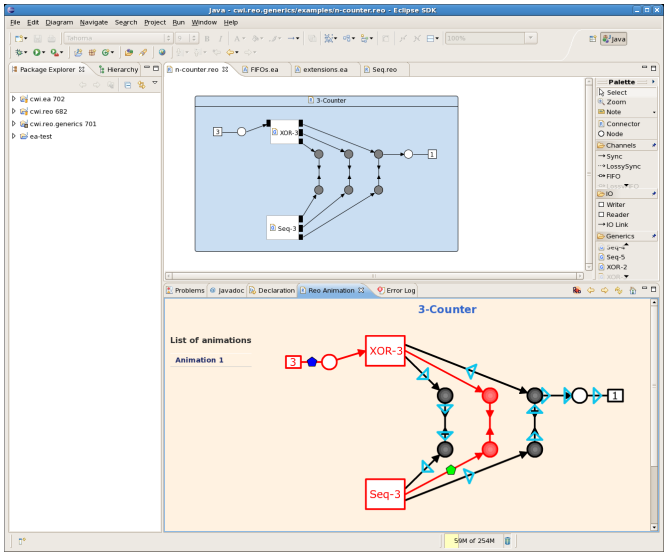


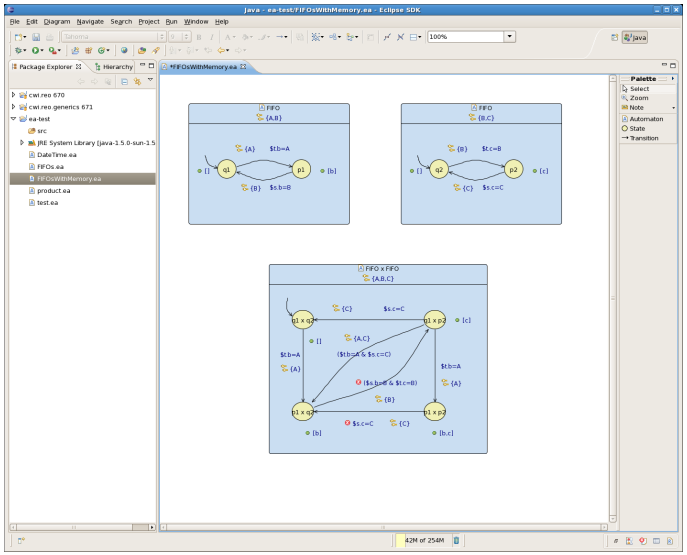
Figure: Graphical Transformation editor.

Outline

Eclipse Coordination Tools (ECT)



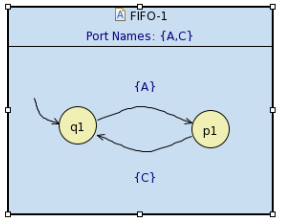
Reo Tools

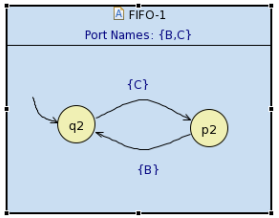


Automata Tools

Automata Tools

- General purpose automata tools.
- Separation of graph structure and attached data (extensions).
- Compositional validation and product.
- Automata extensions are registered using Eclipse' extension point mechanisms.
- Can be (de)activated in the context menu of the editor.



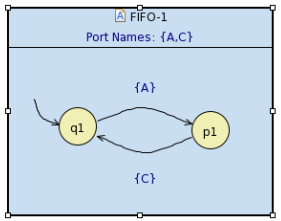


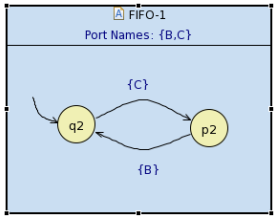
Automata Tools

Currently implemented extensions:

- Start states, port names, local memory, constraints, costs.

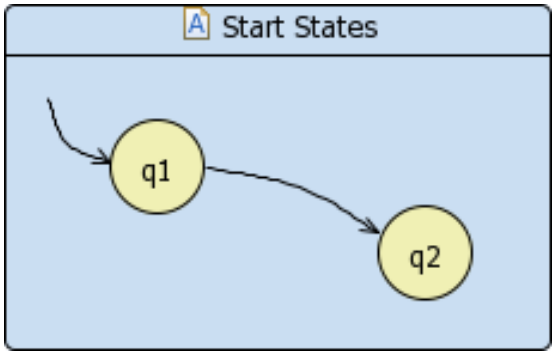
$\Rightarrow$ Constraint automata with memory and costs.





Start State Extensions

- Boolean flag attached to states. It indicates whether a state is a start / initial state.

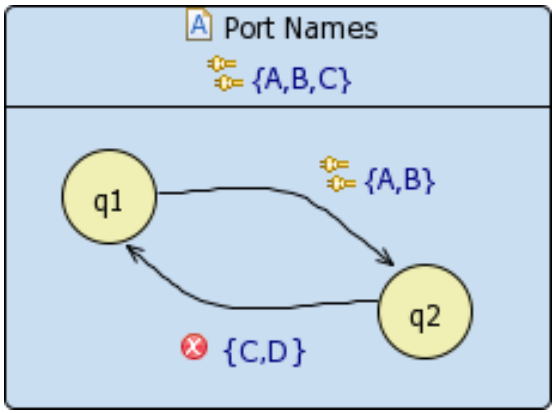


Extension Product

$$isStart_{1\times 2} = isStart_1 \wedge isStart_2$$

Port Names Extensions

- List of port names that is attached to the automaton itself and all of its transitions.
- Validation routines check for illegal (e.g. undefined, duplicate) port names.

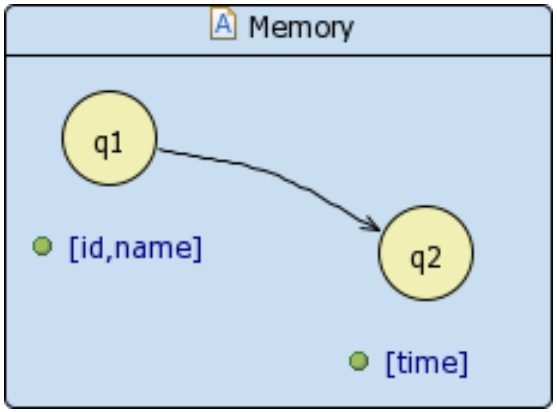


Extension Product

$$
\begin{aligned}
Names_{1\times 2} &= Names_1 \cup Names_2 \\
names_{1\times 2} &= names_1 \cup names_2 \text{ iff} \\
&\quad Names_1 \cap names_2 = Names_2 \cap names_1
\end{aligned}
$$

Memory Extensions

- List of memory cell names that is attached to states.
- Can be referenced in constraints.

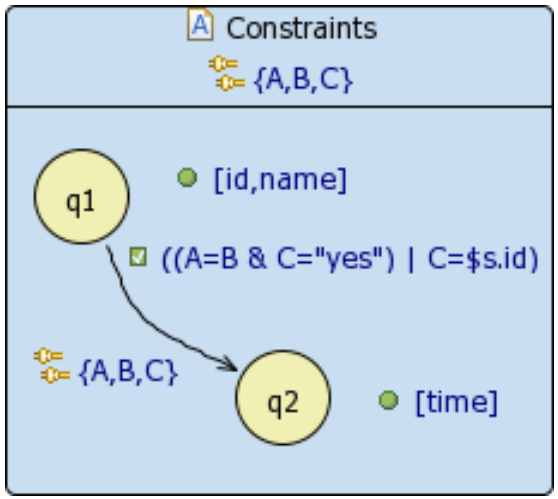


Extension Product

$$memory_{1\times 2} = memory_1 \uplus memory_2$$

Constraint Extensions

- Constraints over the port names and the memory cells are attached to the transitions.
- Only the memory cells of the source and the target state can be used (local memory).

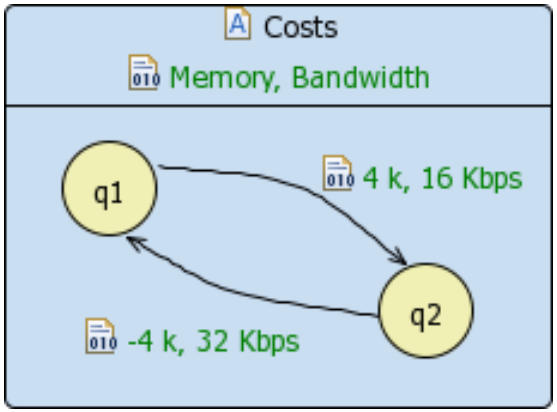


Extension Product

$$constraint_{1\times 2} = constraint_1 \wedge constraint_2$$

Cost Extensions

- List of cost algebras is attached to the automaton.
- List of cost values is attached to the transitions.
- Implementations of the algebras handle the parsing and the product of cost values.



Extension Product

$cost_{1\times 2} = cost_1 \oplus cost_2$

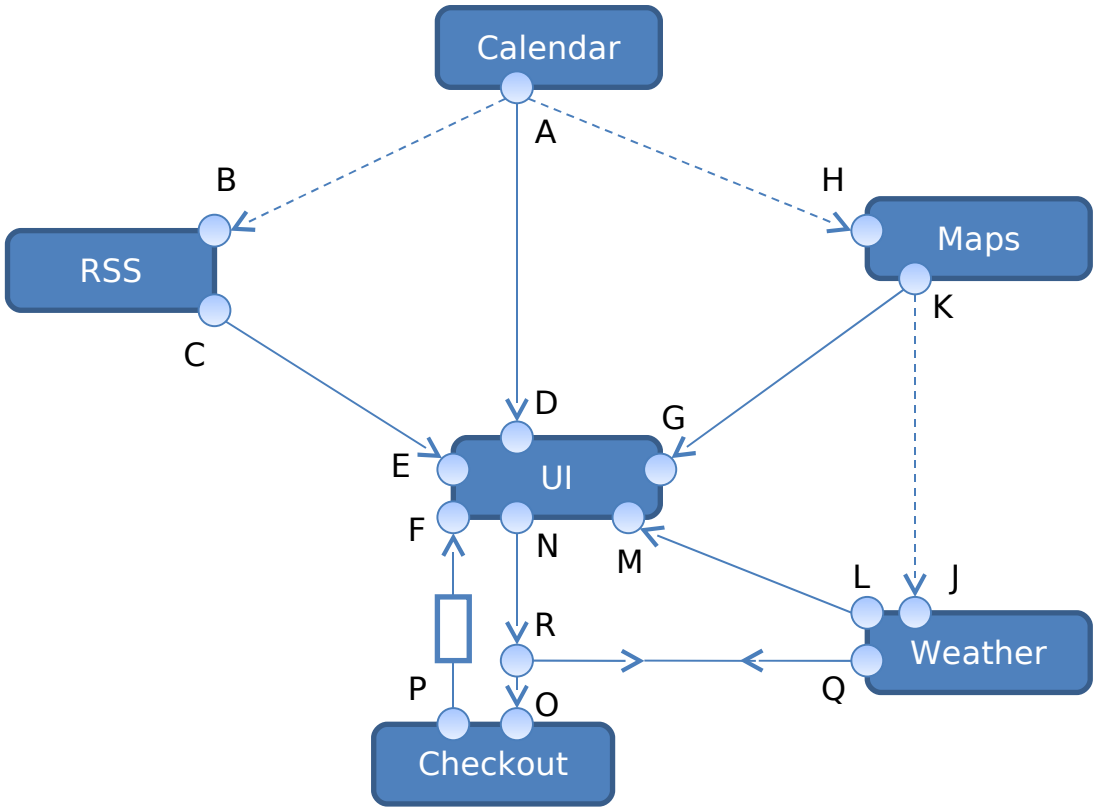
Outline

Reo at Runtime

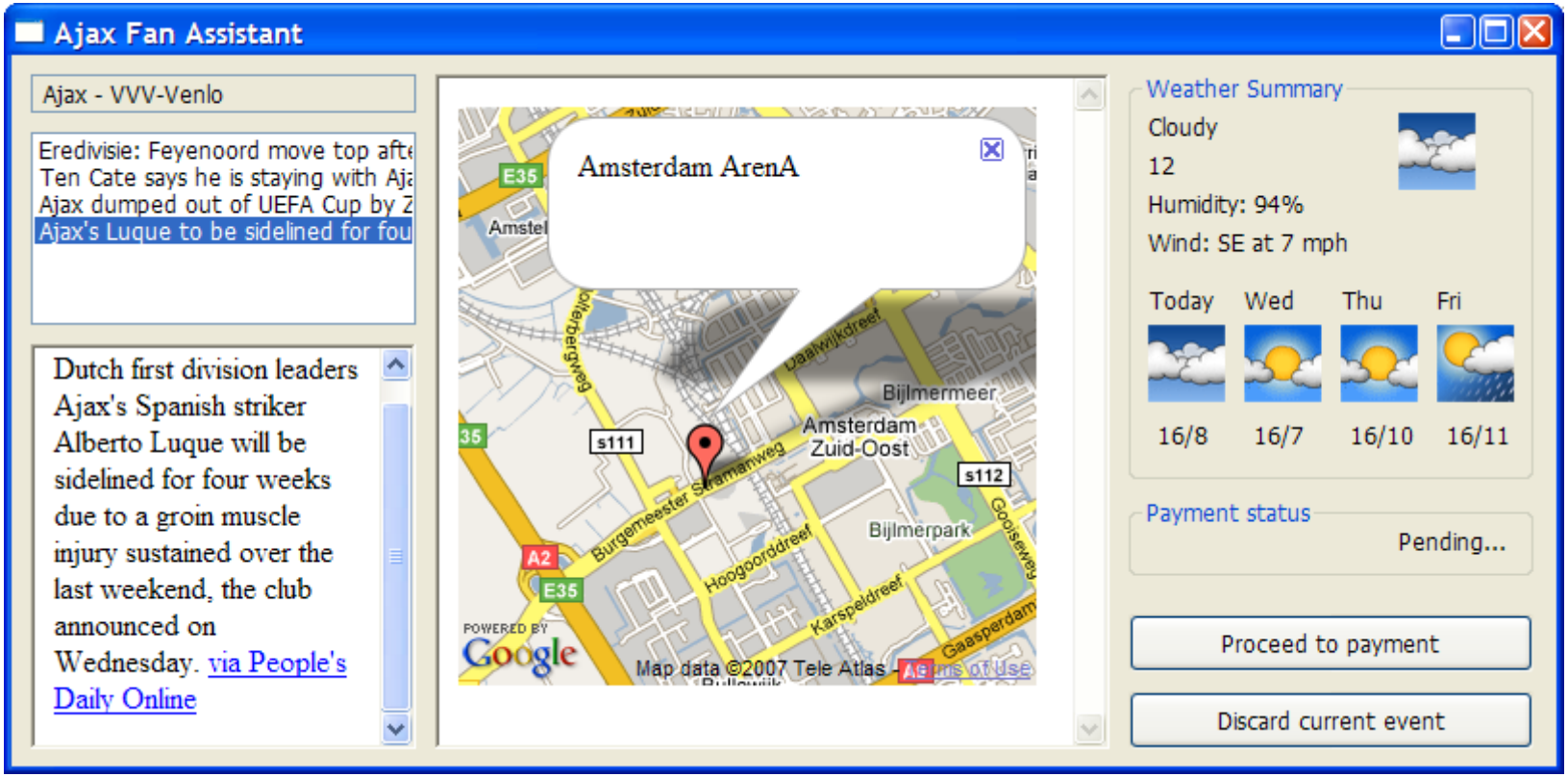
Implementations:

1. Code generation from constraint automata (a.k.a. CASP).
   Reo connectors have to be converted to constraint automata first.
2. Local runtime engine based on constraint programming.
   Adaptive, but centralized implementation.
3. Distributed Reo implemented with Scala Actors.
   Designed to support arbitrary reconfigurations.

Reo at Runtime



Reo at Runtime



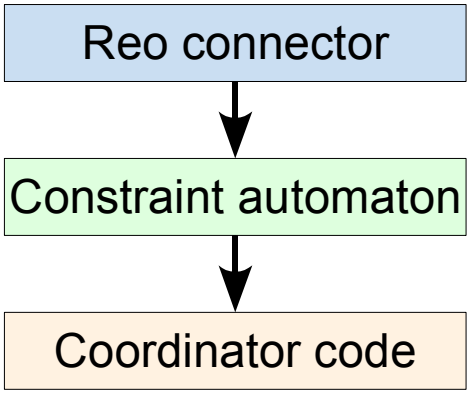
Code Generation

Code generation from constraint automata:

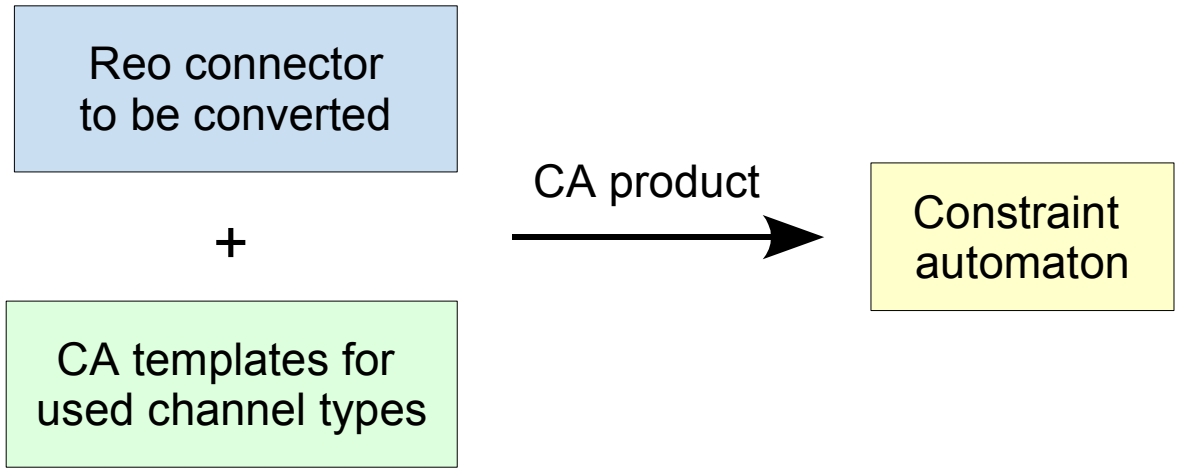
1. Collect (or implement first, if required) component / service wrappers.
2. Design a coordinator as a CA or derive one from a Reo connector.
3. Use graphical wiring tool to attach the wrappers to the coordinator.

Designing the Coordinator

1. Create Reo circuit for collected components.
2. Convert the connector to a CA.
3. Generate coordinator code.



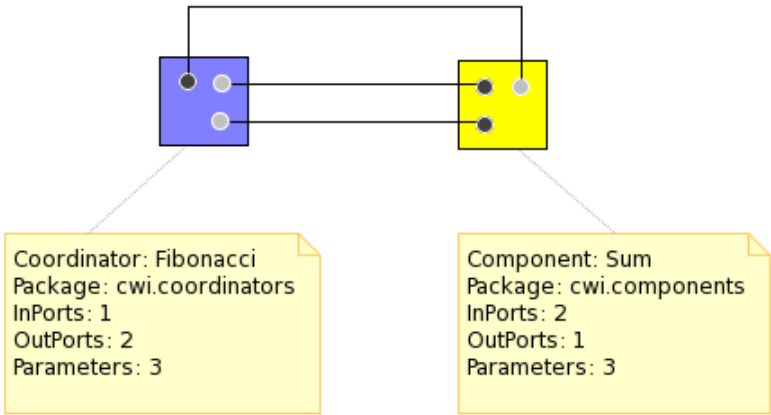
Reo2CA Conversion



- Constraint automata as semantical model for Reo.
- Starting point for model checking and code generation.

Wiring the Components

1. Drag'n'Drop components and coordinators into the editor.
2. Wire the component ports with coordinator ports.
3. Generate runnable application.



Conclusion

What we can do already:

- Design, simulate and model check Reo connectors.
- Derive formal automata models from connectors.
- Generate runnable code from connectors and integrate them with existing component implementations.
- Use predefined wrappers for service protocols (e.g. RSS, SOAP, REST) and specific services (e.g. Google calendar).

Conclusion

What we want to do in the future:

- Modeling of QoS properties (CooPer project).
- Integration with workflow languages and grid architectures (WoMaLaPaDia project).
- Architectural view on services, coordination, grid etc.
- Framework based implementation.

Links

Eclipse Coordination Tools.
http://www.cwi.nl/∼koehler/ect

Reo Animation Repository.
http://www.cwi.nl/∼proenca/webreo