Model Checking Reo Connectors with mCRL2

Natallia Kokash

Introduction

- COMPAS Project
  - Formalization of Business Process Models
  - Formalization of compliance requirements
- Reo
- mCRL2
- Model checking Reo process models with mCRL2
  - Translation from Reo to mCRL2
  - Unified Control and Data Flow Analysis
- Tool support and examples
- Conclusions and Future Work

COMPAS project

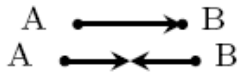
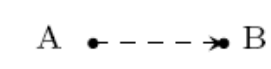
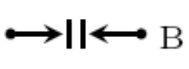
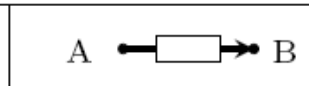
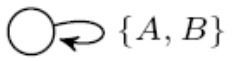
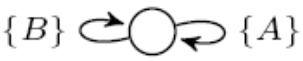
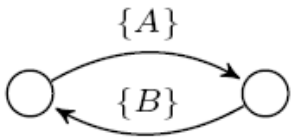
- **COMPAS**
  - Compliance-driven Models, Languages, and Architectures for Services
- **Goal**
  - Ensure dynamic and on-going compliance of software services to business regulations and user requirements
- **Methodology, terminology and research results**
  - http://www.compas-ict.eu/
  - *Compliance* is conformity in fulfilling compliance requirements
  - *Compliance requirement* is a constraint or assertion that results from the interpretation of the compliance sources
  - *Compliance source* is a document that is the origin of compliance requirements (e.g., SOX, HIPAA, license)

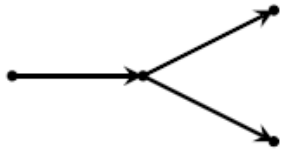
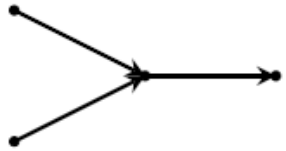
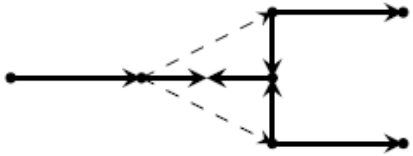
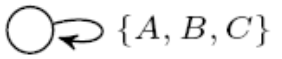
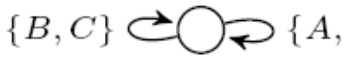
Formal BP analysis in COMPAS

- *Business process* is a composition of activities into a structured order that implements the procedure to be followed in order to achieve a business goal
- *Behavioral model* is a description of how an actor (e.g., stakeholder, service) acts or interacts with other actors
- *Compliance rule* is an operative definition of a compliance requirement
- **Goal:** Check whether compliance rules hold for formal behavioral models of business processes
- *Kokash, N., Arbab, F.: "Formal Behavioral Modeling and Compliance Analysis for Service-Oriented Systems", FMCO'08, vol. 5751 of LNCS, Springer, pp. 21-41*

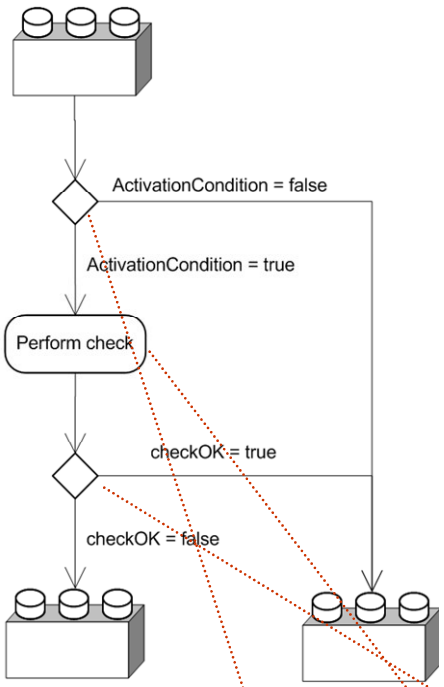
Reo coordination language

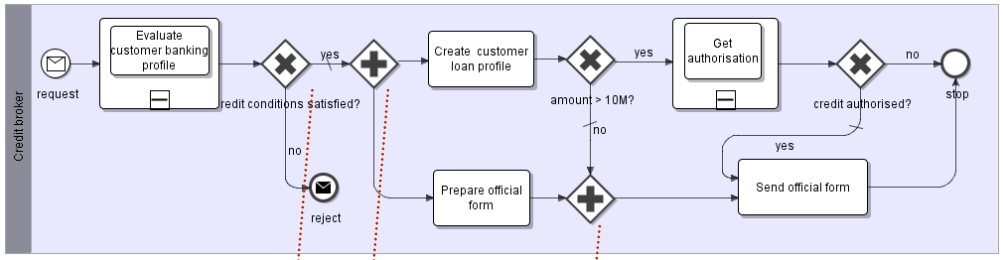
- Components (services) communicate through channels composed to complex connectors
- Semantics:
  - Constraint automata
  - Intentional automata (support for context-dependency)

Sync, SyncDrain	LossySync	AsyncDrain	Fifo1
			
			

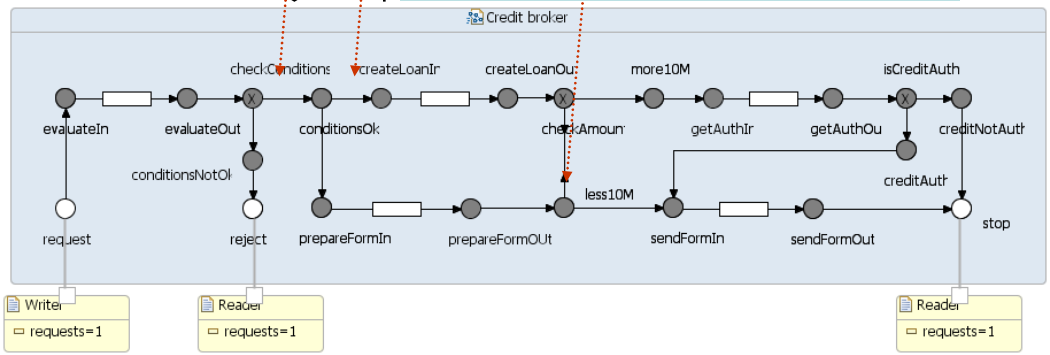
Replicator	Merger	Router
		
		

Formalization of business processes

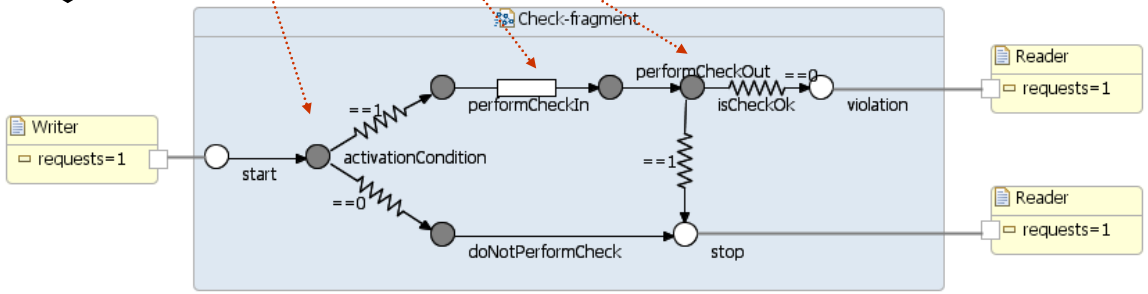




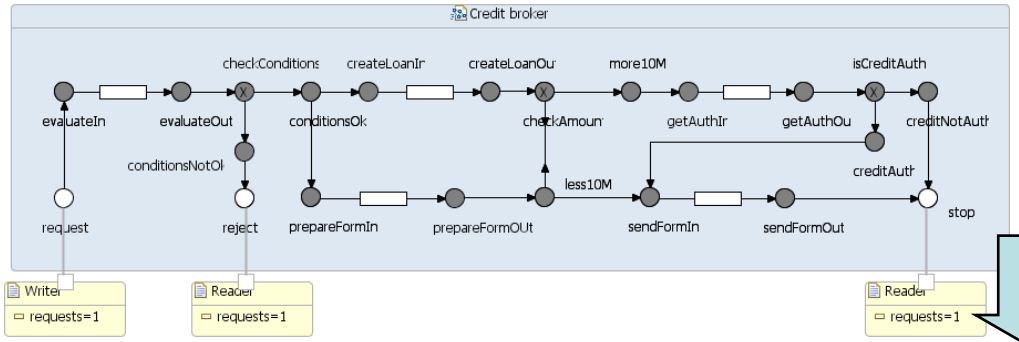
BPMN2Reo converter

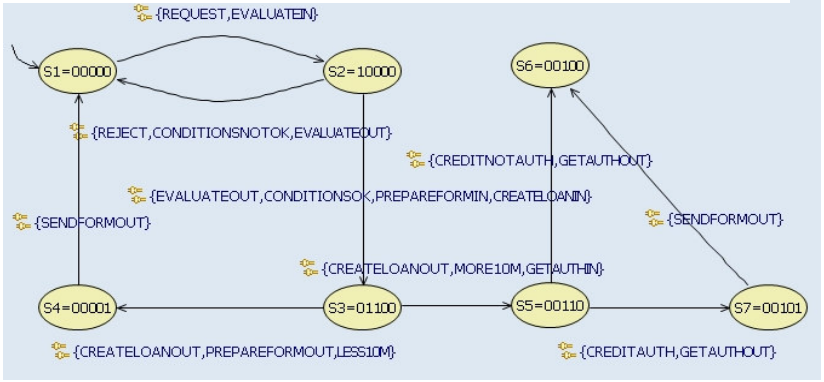


UMLADs2Reo converter

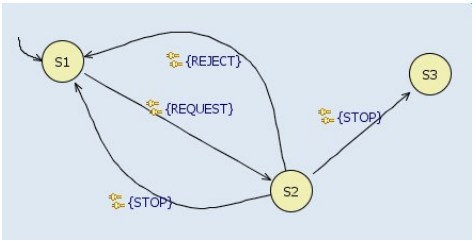


Formalization of business processes





Reo2ExtendedAutomata converters



- Check that admissible states *reject* or *sendFormOut* are eventually reached
  - **G**(*request* → **F** (*reject* U *sendFormOut*)

Business process analysis

- Control flow analysis
  - *Deadlock* is a situation when a service or a process stays idle waiting for resources permanently blocked by another party or messages that will never arrive.
  - *Livelock (starvation of progress)* is a situation when a process or a service is not blocked but doesn't make any progress nevertheless.
  - *Problems with synchronization* conflicts can be introduced by joining fork concurrent paths with a merge structure which in some workflow specification languages (e.g., BPMN) results into unintentional multiple activation of states that follow the merge node.
- Data flow analysis
  - Deadlocks, livelocks and problems with synchronization may appear due to improper conditions on data (e.g., while data-based choice gateway is used in BPMN).
  - Compliance requirements are data dependent.

mCRL2

- Behavioral specification language
- Associated toolset
  - Developed at TU Eindhoven (+ LaQuSo, CWI and Twente University)
- Based on the algebra of communicating processes (ACP)
- Extended with data and time
  - Built-in data types: *Bool, Nat, Pos, Int, Real*
  - Algebraic data types
    - constructors, recognition and projection functions
  - Built-in support for lists, sets and bags
  - User-defined functions (λ calculus)
- Number of industrial case studies
- http://www.mcrl2.org/

mCRL2 specification language

- Actions are atomic events (e.g. a firing of a port or a request arrival in a Reo connector)
- Processes are the active entities defined as expressions over actions and other processes
  - Multiaction: $a|b$ (synchronized actions)
  - Alternative composition: $a + b$ (nondeterministic choice)
  - Sequence composition: $a.b$ (*b started after a*)
  - Conditional: $exp \rightarrow a \lozenge b$ (if-then-else)
  - At operator: $a^c t$ (action a happens at time t)
  - Parallel composition: $a||b$ (interleavings a.b + b.a + a|b)
- Actions and processes can be parametrized with data
  - Summation: $\sum_{d \in D} a(d)$ ($a(d_1) + a(d_2) + a(d_3)\ldots$)

mCRL2 specification language

- Renaming: $\rho_R(a)$ where R is a set of renamings of the form $b \rightarrow c$, meaning that every occurrence of b in a is replaced by c
- Hiding: $\tau_H(a)$ renames all actions of H in a to τ
- Restriction (allow): $\nabla_R(a)$ where R specifies which actions are allowed to occur in a
- Blocking: $\partial_B(a)$ where B is a set of actions that is not allowed to occur in a
- Communication: $\Gamma_C(p)$, where C is a set of allowed communications of the form $a_0|...|a_n \rightarrow c$, $n \geq 1$ which means that every group of actions $a_0|...|a_n$ within a multiaction is replaced by an action c

Example of mCRL2 specification

Dining philosophers

```
eqn K = 2;
map K: Pos;

act get, put, up, down, lock, free: Pos#Pos;
    eat: Pos;

proc
  Phil(n: Pos) = get(n, n) . get(n, if (n == K, 1, n+1)). eat(n). put(n, n)
    . put(n, if (n == K, 1, n+1)) . Phil(n);

  Fork(n: Pos) = sum m: Pos . up(m, n) . down(m, n) . Fork(n);

init allow( { lock, free, eat },
       comm( { get|up->lock, put|down->free },
       Phil(1) || ... || Phil(K) || Fork(1) ||…|| Fork(K) ));
```

Reo to mCRL2 (Constraint automata semantics)

- Data flow observed at a channel end = action
- Synchronous channel, synchronous drain
  - *Sync = A|B.Sync;*
- Non-deterministic synchronous lossy channel
  - *LossySync = (A|B + A).LossySync;*
- Asynchronous drain
  - *AsyncDrain = (A + B).AsyncDrain;*
- FIFO1
  - *FIFO1 = A.B.FIFO1;*
  - *FullFIFO1 = B.FIFO1;*
  - Alternative encoding: *FIFO1(f: Bool) = (¬f → A ◊ B).FIFO1(¬f);*
- Replication node
  - *Replicator = X|Y|Z.Replicator;*
- Merge node
  - *Merger = (X|Z + Y|Z).Merger;*

Reo to mCRL2: Data support

act *A, B: Data*

- Synchronous channel
  - $Sync = \sum_{d \in Data} . A(d)|B(d) . Sync;$
- Synchronous drain
  - $SyncDrain = \sum_{d1,d2 \in Data} . A(d_1)|B(d_2) . SyncDrain;$
- Synchronous lossy channel
  - $LossySync = \sum_{d \in Data} . (A(d)|B(d) + A(d)) . LossySync;$
- Asynchronous drain
  - $AsyncDrain = \sum_{d \in Data} . (A(d) + B(d)) . AsyncDrain;$
- Filter 
  - *Filter = sum* $\sum_{d \in Data} . (exp(d) \rightarrow A(d)|B(d) \lozenge A(d))$. *Filter, where exp(d) is a boolean expression*
- Transformer 
  - $Transformer = \sum_{d \in Data} . A(d)|B(exp(d)) . Transformer;$
- Replication node
  - $Replicator = \sum_{d \in Data} . X(d)|Y(d)|Z(d) . Replicator;$
- Merge node
  - $Merger = \sum_{d \in Data} . (X(d)|Z(d) + Y(d)|Z(d)) . Merger;$

Reo to mCRL2: Data and time support

- FIFO1
  - *DataFIFO1 = struct empty?isEmpty | full(e:Data)?isFull;*
  - *FIFO1(f: DataFIFO1) = $\sum_{d \in Data}$ isEmpty(f)→A(d).Fifo1(full(d)) ◊ B(e(f)).FIFO1(empty))*
- *T*-timer with off- and reset- options
  - Reacts differently to different data inputs:
    - *DataTimer = struct reset?isReset | off?isOff | timeout | other(e: Data)?isOther*
  - Has two states
    - *State = struct OFF?isOFF | ON?isON*
  - State *s* (timer ON or OFF), current time *x*, timer delay *t*
    - *Timer(s: State, x: Real, t: Real) =*
      isOFF(s)→$\sum_{d \in DataTimer}$ isOther(d)→A(d).Timer(ON, 0, t) +
      isON (s) → ((x < t) → $\sum_{d \in DataTimer}$
      isReset(d) → A(d).Timer(ON, 0, t) +
      isOff (d) → A(d).Timer(OFF, x, t) +
      tick^{c}x.Timer(ON, x + 1, t))
      ◊ B(timeout).Timer(OFF, x, t)

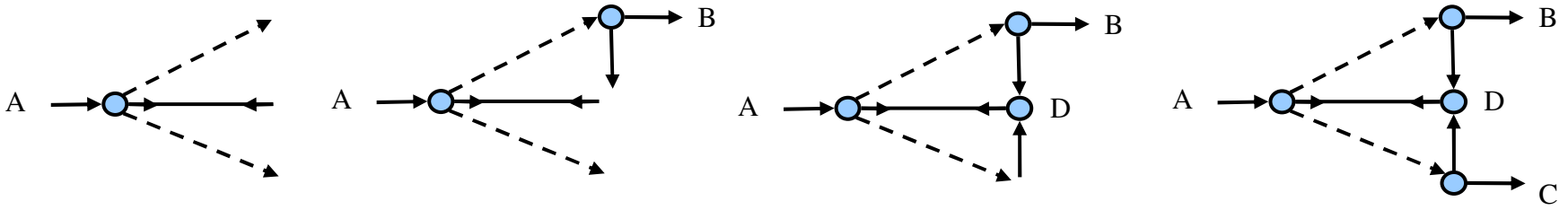
Reo to mCRL2: Global data types

- A connector should deal with any data items consumed by its source nodes
- Given a set of elementary data types $DT_1,\ldots,DT_n$ (e.g., inferred from web service interface specifications), the global data type is described as follows:
  - $Data = struct\ D_1(e_1: DT_1)|\ldots|D_n(e_1: DTn)$
- *Join node*
  - $Join = \sum_{d1,d2 \in Data}. (X(d_1)|Y(d_2)|Z(tuple(d_1, d_2)).Join;$
- For *m*-join node $tuple(e_1: Data, e_2: Data,\ldots, e_m: Data)$ is added to the Data description, e.g.,
  - $Data = struct\ D_1(e_1: DT_1)|\ldots|D_n(e_1: DT_n)\ |tuple(e_1: Data, e_2: Data)$
- **Note:** expressions for filter and transformer channels become dependent on the structure of the Reo connector

Reo to mCRL2: Composition

Synchronize and hide actions corresponding to the connected channels

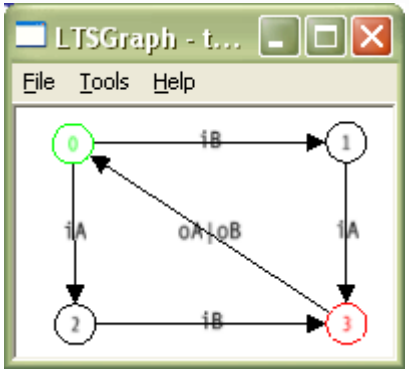
- **Problem**
  - Reduce the size of the state space while building the LTS for the mCRL2 specification of a Reo connector
- **Idea**
  - Iterated connector construction

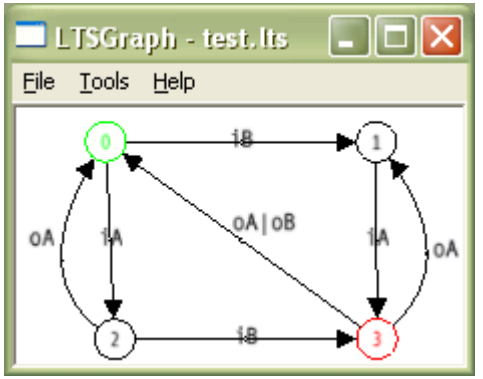


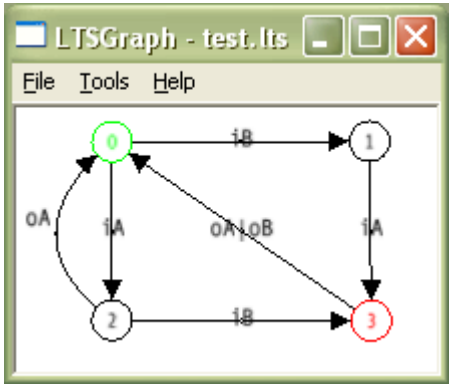
1. $P_0 = \partial_{\text{ends of connected channels}}(\Gamma_{\text{handshaking at Node1}}(Node_1 \,||\, Sync_1 \,||\, LossySync_1 \,||\, LossySync_2 \,||\, SyncDrain_1))$
2. $P_1 = \partial_{\text{ends of connected channels}}(\Gamma_{\text{handshaking at Node2}}(Node2 \,||\, Sync_2 \,||\, Sync_3 \,||\, P_0))$
3. $P_2 = \partial_{\text{ends of connected channels}}(\Gamma_{\text{handshaking at Node3}}(Node3 \,||\, Sync_4 \,||\, P_1))$
4. $P_3 = \partial_{\text{ends of connected channels}}(\Gamma_{\text{handshaking at Node4}}(Node4 \,||\, Sync_5 \,||\, P_2))$

Reo to mCRL2 (Intentional automata semantics)

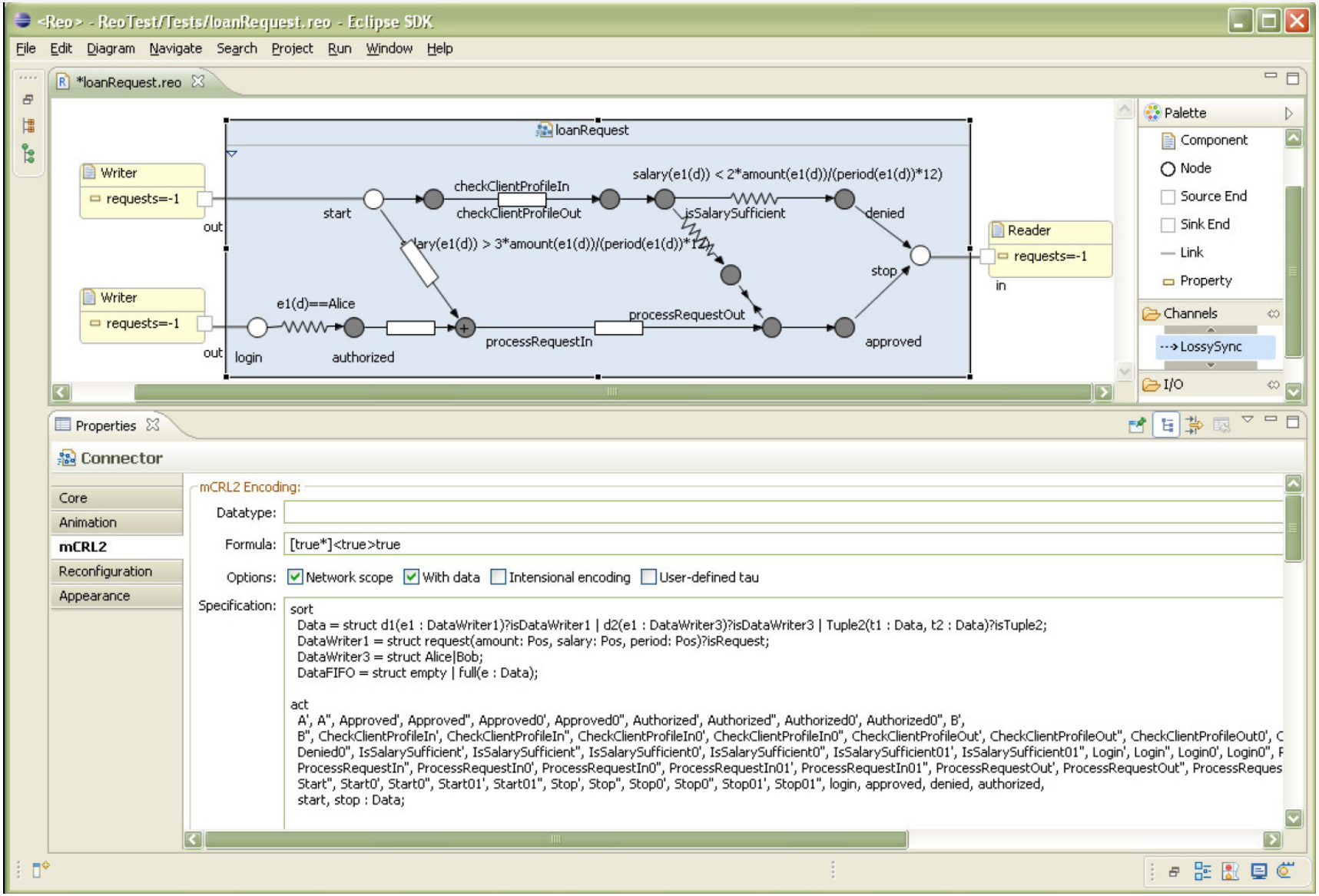
- Two types of actions:
  - A channel end is ready to write/read *(iA)*
  - Data flow observed at a channel end *(oA)*
- Synchronous channel, synchronous drain
  - *Sync(a: Bool, b: Bool) =*
    ¬a → iA.Sync(¬a, b) +
    ¬b → iB.Sync(a, ¬ b) +
    (a ∧ b) → oA|oB.Sync(¬a, ¬b);
- Context independent lossy channel
  - *LossySync(a: Bool, b: Bool) =*
    ¬a → iA. LossySync(¬a, b) +
    ¬b → iB. LossySync(a, ¬ b) +
    a → (b → oA|oB. LossySync(¬ a, ¬b) +
    oA. LossySync (¬a, b));
- Context dependent lossy channel
  - *LossySync(a: Bool, b: Bool) =*
    ¬a → iA. LossySync(¬a, b) +
    ¬b → iB. LossySync(a, ¬ b) +
    a → (b → oA|oB. LossySync(¬ a, ¬b) +
    ¬b → oA. LossySync (¬a, b));



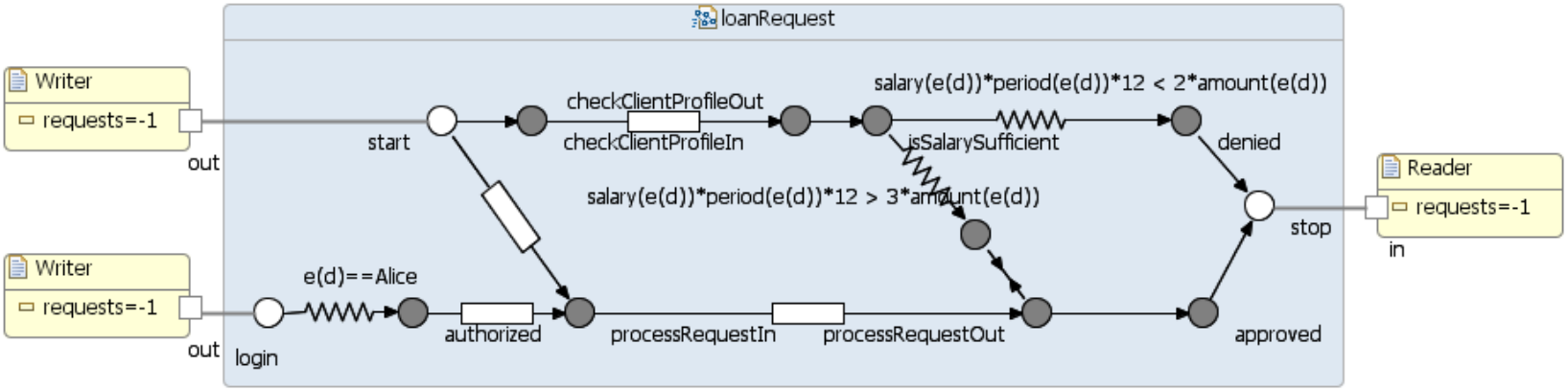




Tool support

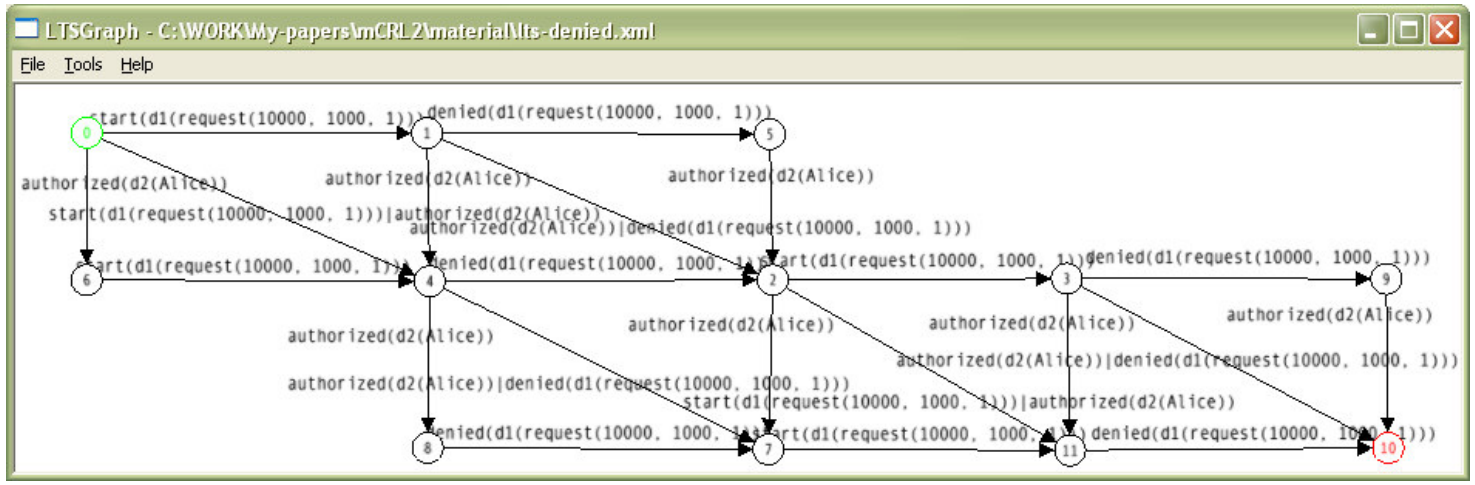


Data-flow analysis with mCRL2

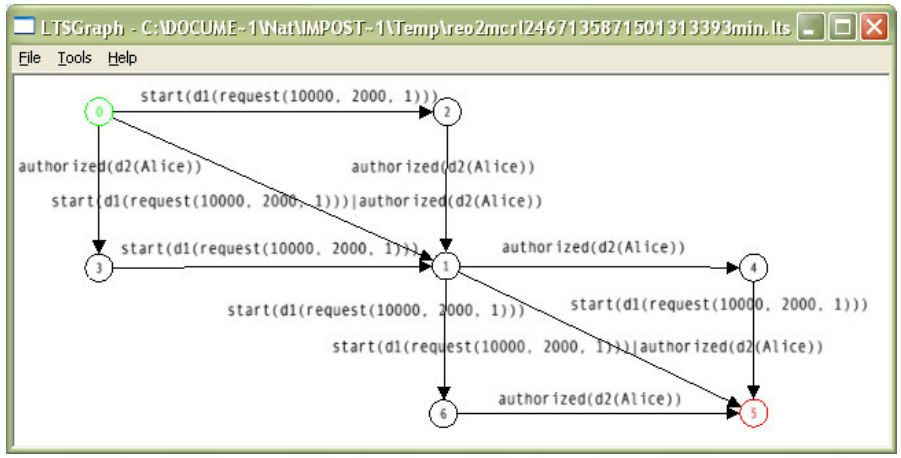


- μ calculus (property specification format for mCRL2)
  - No deadlock:
    - [true*]<true>true
  - No livelock:
    - [true*]mu X.[tau]X
  - Loan request is always denied or approved:
    - Data agnostic:
      - nu X.[true]X && mu Y.([!(denied || approved)]Y && <true>true)
    - Data aware:
      - nu X.[true]X && forall d:Data. mu Y.([!(denied(d) || approved(d))]Y && <true>true)

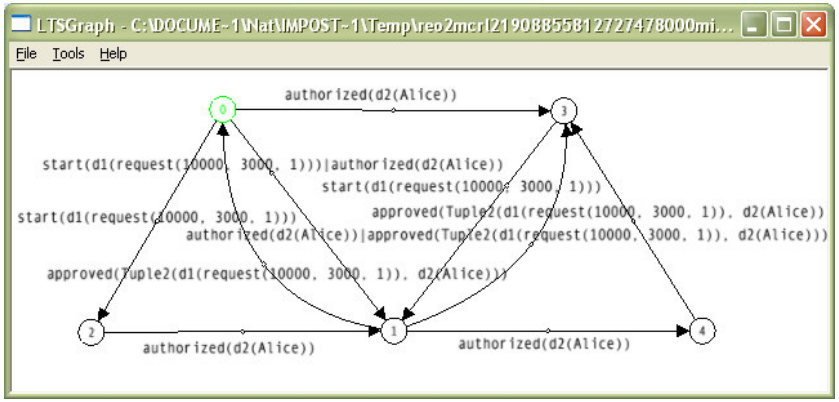
Loan request scenario: different instances



Salary = 1000



Salary = 2000



Salary = 3000

Comparison of model checking tools for Reo

- Vereofy (University of Dresden)
  - **Advantages:**
    - Developed for Reo and Constraint Automata
    - Counterexamples
  - **Disadvantages:**
    - No support for abstract data types
    - Global domain for all components
    - Primitive data constraint specification language (for filter channels)
- mCRL2 toolset
  - **Advantages:**
    - Powerful support for data
    - Very rich property specification format (μ calculus)
  - **Disadvantages:**
    - Hard to extract counterexamples
    - For infinite domains model checker often does not terminate (problems with algorithms for formulae rewriting)
    - Inability to define some useful data domains
    - Intentional model: state explosion problem
- CADP toolset (INRIA)
  - **Advantages:**
    - Fully compatible with mCRL2
    - Many useful tools (e.g., for performance evaluation)
  - **Disadvantages:**
    - License

Conclusions

- Full featured model checking of Reo connectors
  - Control + data flow analysis
  - Abstract data types
  - Complete tool support
    - Automated generation of mCRL2 code from graphical models
    - First implemented plug-in that deals with
      - Filter and transformer channels
      - Intentional semantics
      - Time channels
- Relation between Reo and ACP
- *Eclipse Coordination Tools* are an interesting toolset for business process and service composition analysis
  - Good alternative to Petri nets
  - Better fits service-oriented computing paradigm
  - *N. Kokash, C. Krause, E. de Vink: "Data-aware Design and Verification of Service Compositions with Reo and mCRL2", ACM SAC 2010, Technical track on Service-Oriented Architectures and Programming (SOAP)*

Future Work

- Work on tools usability
  - Analysis across several connectors
  - Obtaining data types from WSDL specifications
  - Other verification tools (e.g., CADP, simulators)
- Timed Reo
  - What properties can be checked with mCRL2
- Application to COMPAS case studies
  - Express compliance requirements and common BP properties in μ-calculus
  - **Problem:** property specification language is very expressive, but hard to use