Modeling and Verification of Connectors in Complex Systems

Sun Meng

LMAM & Department of Information Science, School of Mathematical Sciences
Peking University
http://www.math.pku.edu.cn/teachers/sunm

Thanks to: B. K. Aichernig (TUG), F. Arbab (CWI), L. Aștefănoaei (INRIA), C. Baier (TUD), L. Barbosa (UM), F. de Boer (CWI), T. Chothia (Birmingham), N. Kokash (CWI), M. Kwiatkowska (Oxford), Y. Li (PKU), Y.-J. Moon (INRIA), H. Qu (Oxford), J. Rutten (CWI), R. van der Mei (VUA), C. Verhoef (CWI)

Workshop on Probabilistic and Hybrid System Verification, Beijing, September 26, 2013
Outline

• Coordination in complex systems
• Reo and Eclipse Coordination Tools
• Synthesis of connectors from BPMN / UML models
• Verification and Performance Analysis for connectors
• Conclusion and future work
Sources of Complexity in Systems

• Complexity inherent in task/algorithm/computation
  • Examples:
    • Computations/equations in quantum mechanics, astronomy, engineering, etc.
    • Bit-map to jpeg conversion, sorting, etc.
  • This type of complexity is not bewildering!
    • Many good, intricate mathematical models have been developed to tame the complexity.

• Complexity arising from composition of simple components
  • Example:

    • Bewildering complexity emerges out of interaction
    • Good formal models to tame this complexity?
Models of Concurrency

- Traditional models are action based
  - Petri nets
  - Work flow / Data flow
  - Process algebra / calculi
  - Actor models / Agents
  - ...

- In prominent models, a system is composed from building blocks that represent actions/processes

- Interaction becomes an implicit side-effect
  - Makes coordination of interactions more difficult to
    - Specify
    - Verify
    - Manipulate
    - Reuse
Interaction Based Concurrency

• Start with a set of primitive interactions as binary constraints
• Define (constraint) composition operators to combine interactions into more complex interactions
• Properties of the resulting model of concurrency depend on
  • Set of primitive interactions
  • Composition operators
• As constraints, interaction protocols can be manifested independently of the processes that they engage
  • Connectors
• Imposing an interaction on actors exogenously coordinates their activities
Exogenous Coordination

• P and C are **black-box** components that:
  • Offer no inter-components methods nor make such calls
  • Do not send/receive targeted messages
  • Their only means of communication is through **blocking I/O** primitives that they can perform on their own **ports**.
  • Composing P and C with different **connectors** (that impose different **protocols from outside**) constructs different systems.

\[ \text{Modeling and Verifying Connectors} \]
Reo: An Exogenous Coordination Language

• Reo is an exogenous coordination language for compositional construction of interaction protocols.

• Interaction is the only first-class concept in Reo:
  • Explicit constructs representing interaction
  • Composition operators over interaction constructs

• A (coordination or interaction) protocol:
  • manifests as a connector
  • gets imposed on its engaged components/services from outside
  • remains mutually oblivious to its engaged components/services

• Reo offers:
  • Loose(st) coupling
  • Arbitrary mix of asynchrony, synchrony, and exclusion
  • Open-ended user-defined primitive channels
  • Distribution and mobility
  • Dynamically reconfigurable connectors

• http://reo.project.cwi.nl
Reo: A Coordination Language

/* every 5 sec */
produce(text)
| synchronize
| if (display.isReady(RED))
| display.show(RED,text)

/* every 3 sec */
produce(text)
| synchronize
| if (display.isReady(GREEN))
| display.show(GREEN,text)

bufferRED := Empty
bufferGREEN := Empty

isReady (from)
| if (from = RED)
| bufferRED = Empty
| if (from = GREEN)
| bufferGREEN = Empty

show (from,text)
| if (from = RED)
| bufferRED := text
| if (from = GREEN)
| bufferGREEN := text
| if (bufferRED isnot empty and
| bufferGREEN isnot empty)
| displayText(bufferRED)
| displayText(bufferGREEN)
| bufferRED = Empty
| bufferGREEN := Empty
Reo: A Coordination Language
Channels

• Atomic connectors in Reo are called *channels*.
• Reo generalizes the common notion of channel.
• A **channel** is an abstract communication medium with:
  • exactly **two ends**; and
  • a **constraint** that relates (the flows of data at) its ends.
• Two types of channel ends
  • **Source**: data enters into the channel.
  • **Sink**: data leaves the channel.
• A channel can have two sources or two sinks.
• A channel represents a **primitive interaction**.
Reo Connectors

- FIFO1 channel
- Synchronous channel
- Lossy synchronous channel
- Filter channel
- P-producer
- Synchronous drain
- Asynchronous drain
- Synchronous spout
- Asynchronous spout
- Timer channel

Exclusive choice (deffered XOR)

Valve connector: controls flow from A to B
Eclipse Coordination Tools

• A set of Eclipse plug-ins provide the ECT visual programming environment.
• Protocols can be designed by composing Reo circuits in a graphical editor.
• The Reo circuit can be animated in ECT.
• ECT can automatically generate the CA for a Reo circuit.
• Model-checkers integrated in ECT can be used to verify the correctness properties of a protocol.
• ECT can generate executable (Java/C) code from a CA as a single sequential thread.
• http://reo.project.cwi.nl
# Eclipse Coordination Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reo graphical editor</td>
<td>Drag and drop editing of Reo circuits</td>
</tr>
<tr>
<td>Reo animation plug-in</td>
<td>Flash animation of data-flow in Reo circuits</td>
</tr>
<tr>
<td>Extensible Automata editor and tools</td>
<td>Graphical editor and other automata tools</td>
</tr>
<tr>
<td>Reo to constraint automata converter</td>
<td>Conversion of Reo to Constraint Automata</td>
</tr>
<tr>
<td>Verification tools</td>
<td>• Vereofy model checker (<a href="http://www.vereofy.de">www.vereofy.de</a>)</td>
</tr>
<tr>
<td></td>
<td>• mCRL model checking</td>
</tr>
<tr>
<td></td>
<td>• Bounded model checking of Timed Constraint Automata</td>
</tr>
<tr>
<td>Java code generation plug-in</td>
<td>State machine based coordinator code (Java, C, and CA interpreter for Tomcat servlets)</td>
</tr>
<tr>
<td>Distributed Reo middleware</td>
<td>Distributed Reo code generated in Scala (Actor-based Java)</td>
</tr>
<tr>
<td>(UML / BPMN / BPEL) GMT to Reo converter</td>
<td>Automatic translation of UML SD / BPMN / BPEL to Reo</td>
</tr>
<tr>
<td>Algebraic Graph Transformation</td>
<td>Dynamic reconfiguration of Reo circuits</td>
</tr>
<tr>
<td>Markov chain generator (Reo2MC)</td>
<td>Compositional QoS model based on Reo Analysis using, e.g., probabilistic symbolic model checker Prism (<a href="http://www.prismmodelchecker.org">http://www.prismmodelchecker.org</a>)</td>
</tr>
</tbody>
</table>

...
Tool Snapshot

Modeling and Verifying Connectors
Reo graphical editor

Reo simulation plug-in

Tool Snapshot

Reo to constraint automata converter
Snapshot of Reo Editor

Modeling and Verifying Connectors
Reo Animation Tool
Constraint Automata Tools

• ECT includes a graphical editor for CA and related automata models
  • Create and edit automata graphically
  • Perform product and hiding on automata
• ECT includes tools to automatically derive the CA of a Reo circuit
• ECT includes simulator engines to show automata runs
Constraint Automata Editor
Synthesis from BPMN to Reo

Input of BPMN-to-Reo Converter
Output of BPMN-to-Reo Converter
Synthesis from UML SD to Reo

- Sequencers are derived for individual participants
Synthesis from UML SD to Reo

- Nodes for different lifelines are connected pairwise by synchronous or asynchronous channels according to the types and order of messages.
Synthesis from UML SD to Reo

- Reo circuits are structured inductively according to the operators in UML SDs.

- Correctness of the approach is proved by coinduction.
SD-to-Reo Converter

• Accepts UML 2.x SD models as input
• Generates Reo circuits representing the communication protocol
• Can combine SDs for different scenarios and use-cases
• Enables verification and reasoning about the combined protocol
• Originally, a stand-alone tool
• Modified and improved to accept Bouml XMI input
• Support for Eclipse UML2 tool coming
UML SD Editor
SD-to-Reo Converter
References


Verification

• Connectors as designs for refinement checking and test case generation
• Vereofy: Model checker for Reo built in TU-Dresden:
  • Symbolic model, LTL, and CTL-like logic for specification
  • Can also verify properties such as deadlock-freeness and behavioral equivalence
• SAT-based bounded model checking of Timed Constraint Automata
• Translation of Reo to mCRL2 for model checking
• Translation of Reo to Coq for proving properties
Connectors as Designs

• Every connector $R$ can be represented as

$$P(in_R) \vdash Q(in_R, out_R)$$

• $P(in_R)$ ($Q(in_R, out_R)$) is the pre-condition (post-condition) that should be satisfied by inputs $in_R$ (outputs $out_R$) on the source (sink) nodes of $R$.

• $in_R$ and $out_R$ are mappings from sets of source and sink node names of $R$ to timed data streams respectively.
Connectors as Designs

• Implication of predicates establishes a refinement order over connectors. More concrete implementations imply more abstract specifications.

• For two connectors

\[
\text{con: } R_i \langle \text{in: } in_{R_i}, \text{out: } out_{R_i} \rangle \\
\text{pre: } P_i(\text{in}_{R_i}) \\
\text{post: } Q_i(\text{in}_{R_i}, \text{out}_{R_i})
\]

where \( i = 1,2 \), if \( \text{in}_{R_1} = \text{in}_{R_2} \) and \( \text{out}_{R_1} = \text{out}_{R_2} \), then

\[
R_1 \sqsubseteq R_2 = \text{df} (P_1 \Rightarrow P_2) \land (P_1 \land Q_2 \Rightarrow Q_1)
\]

• Pre-conditions on inputs of connectors are weakened under refinement, and post-conditions on outputs of connectors are strengthened.
Connectors as Designs

```
testRefinement of
connector ABCa = R(( a , sa ) : ( c , sc ), ( b , sb )), D< sa > |-D< comp(dc,tc) > and tc =t between2(ta) and D< comp(db,tb) > and tb =t between2(ta) and db =d dc and da =d db

and

connector ABCc = R(( a , sa ) : ( b , sb ), ( c , sc )), D< comp(da,ta) > |-D< comp(db,tb) > and between2(ta) =t tb and da =d db and D< comp(dc,tc) > and between2(ta) =t tc and da =d dc

is: True() and True()
```
References


Vereofy Model Checker

• Symbolic model checker for Reo:
  • Based on constraint automata
  • Developed at the University of Dresden
  • LTL and CTL-like logic for property specification

• Modal formulae
  • Branching time temporal logic:
    • $\text{AG}[\text{EX}[\text{true}]]$
    • check for deadlocks
  • Linear temporal logics:
    • $\text{G}(\text{request} \rightarrow \text{F}(\text{reject} \cup \text{sendFormOut}))$
    • check that admissible states reject or sendFormOut are reached

• [http://www.vereofy.de](http://www.vereofy.de)
Verification with Vereofy

- Modal formulae
  - Branching time temporal logic: $\text{AG}[\text{EX}[\text{true}]]$ – check for deadlocks
  - Linear temporal logics: $\text{G}(\text{request} \rightarrow \text{F}(\text{reject} \cup \text{sendFormOut}))$ – check that admissible states $\text{reject}$ or $\text{sendFormOut}$ are reached
Data-Dependent Control-Flow

• Input parameters:
  • Activation condition
    • Data: b: Boolean
    • Filter condition: b==true, b==false
  • Check condition
    • Data: x, y: Real; (e.g., credit amount, maximal amount)
    • Filter condition: x < y

• Problems:
  • Data constraint specification language is needed
  • Properties that include conditions:
    • G [(b & !(x < y)) → F violation]
Verification with mCRL2

- mCRL2 behavioral specification language and associated toolset developed at TU Eindhoven
  - [http://www.mcrl2.org](http://www.mcrl2.org)
  - Based on the Algebra of Communicating Processes (ACP)
  - Extended with data and time
  - Expressive property specification language ($\mu$ calculus)
  - Abstract data types, functional language ($\lambda$ calculus)
- Automated mapping from Reo to mCRL2
Verification with Coq

Application Layer
- Calculation
- Proof

Component Layer
- DataInput
- Simulate
- DataOutput

Data>Type Layer
- Environment
- DataStreamMatrix

Coq Proof Assistant
Performance Analysis

• Quantitative Intentional Automata (QIA) extend CA with quantitative properties:
  • arrival rates at ports
  • average delays of data-flows between ports

• Quantified Reo circuits are converted to QIA

• Markov Chain models are derived from QIA
  • Resulting Markov Chains are very compact: efficient model checking

• PRISM is used for analysis of MC models
Performance Analysis

- Reo Circuit
- Stochastic Information

Stochastic Reo → QIA Generator → QIA (XML) → QIA2MC → MC (XML) → GMF

Files → Parser → GMF

PRISM, MATLAB, Maple → Graphical Representation
Reo Primitives with Delays

Diagram illustrating Reo primitives with delays.

- A
- B
- dAF
- dFB
- dAB
- dA
- dB
- dALost
- dA
- dB
QIA for FIFO1
QIA for Sync
QIA for LossySync

![Diagram of QIA for LossySync](image-url)
QIA for SyncDrain
QIA of Alternator Reo Circuit
Markov Chain for Alternator
Experiment
Conclusion & Future Work

• Making interaction explicit in concurrency allows its direct
  • Specification
  • composition
  • Analysis
  • Verification
  • reuse
• Reo is a simple, rich, versatile, and surprisingly expressive language for compositional construction of pure (coordination or concurrency) protocols.
• Extension of the language for hybrid systems and related tools development.
Thanks!
Heel hartelijk bedankt!
ich danke Ihnen sehr!
谢谢！
Questions?
Je hebt een problem?
Sie haben ein Problem?
问题？