Model Testing Asynchronously Communicating Objects using Rewriting Modulo AC

Olaf Owe[1], Martin Steffen[1], and Arild B. Torjusen[1,2]

University of Oslo

Norsk Regnesentral

MBT'10, $\Pi\alpha\phi\omega\sigma$

Structure

Background

- Project:
  - modelling asynchronously communicating components in open envqironments
  - object-oriented
  - behavioral interface descriptions
  - automated verification and testing techniques

- Challenges
  - asynchronicity $\Rightarrow$non-determinism $\Rightarrow$state space explosion.

- Approach:
  - tackle complexity by “divide-and-conquer”
  - black-box behavior given by interactions at the *interface*

General setting

Goal: Test components under environment assumptions/schedulings

Approach: Specification language over communication labels

- input interactions: environment *assumptions*.
- output interactions: *commitments* of the component.

$\Rightarrow$ expected observable output behavior under the *assumption* of a certain scheduling of input.

Method: Specification simulates environment behavior.

- execute component and specification in parallel
- *generate* incoming communication from specification
- *test* actual outgoing communication from the component

Main contributions

1. Theoretical basis:
   - formalization of the interface behavior of an asynchronous OO modelling language.
2. Framework for scheduling asynchronous testing of objects.
   - executable specification language
   - method for composing specifications and components under test
   - implementation of a test framework
3. Use Maude's rewriting modulo AC to test only up to observational equivalence.
4. Use Maude's *search* for state exploration (rewriting modulo AC).
5. Experimental results, comparing:
   - modulo AC rewriting.
   - explicit reordering of output events.

Creol

Creol (`www.uio.no/~creol`): object-oriented modelling language for distributed systems

- model distributed systems at a *high level of abstraction*.
- strongly typed, formal operational semantics in rewriting logic
- active concurrent objects
- communication by asynchronous method calls.
- Creol object: acts as a monitor.
- cooperative scheduling, i.e., explicit and conditional release/yields etc.
- *non-deterministic* selection of suspended processes and incoming calls.

Abstract syntax

C	$::=$	$\mathbf{0} \mid C \parallel C \mid \underline{\nu(n{:}T).C} \mid c[\![F, M]\!] \mid \underline{o[c, F, L]} \mid \underline{n\langle t \rangle}$	component
F	$::=$	$l = f, \ldots, l = f$	fields
M	$::=$	$l = m, \ldots, l = m$	method suite
m	$::=$	$\varsigma(n{:}T).\lambda(x{:}T, \ldots, x{:}T).t$	method
f	$::=$	$\varsigma(n{:}T).\lambda().v \mid \varsigma(n{:}T).\lambda().\bot_{n'}$	field
t	$::=$	$v \mid \mathsf{stop} \mid \mathsf{let}\, x{:}T = e\, \mathsf{in}\, t$	thread
e	$::=$	$t \mid \mathsf{if}\, v = v\, \mathsf{then}\, e\, \mathsf{else}\, e \mid \mathsf{if}\, \mathit{undef}(v.l())\, \mathsf{then}\, e\, \mathsf{else}\, e$	expr.
	$\mid$	$v@l(\vec{v}) \mid \underline{v.l(\vec{v})} \mid v.l() \mid v.l := \varsigma(s{:}T).\lambda().v$	
	$\mid$	$\mathsf{new}\, n \mid \mathsf{claim}@(n, n) \mid \underline{\mathsf{get}@n} \mid \mathsf{suspend}(n) \mid \underline{\mathsf{grab}(n)} \mid \underline{\mathsf{release}(n)}$	
v	$::=$	$x \mid n \mid ()$	values
L	$::=$	$\bot \mid \top$	lock status

- *component*: classes, objects, and (named) threads.
- *active*, executing entities: *named threads* $n\langle t \rangle$
- hiding and dynamic scoping: ν-operator

Interface interactions

- Steps occurring at the interface.
- Component/environment: exchange information via *call*- and *return*-labels:

$$\begin{array}{lcll} \gamma & ::= & n\langle call\ n.I(\vec{v})\rangle \mid n\langle \mathrm{return}(n)\rangle \mid \nu(n{:}T).\gamma & \text{basic labels} \\ a & ::= & \gamma? \mid \gamma! & \text{input and output labels} \end{array}$$

- External steps

$$\Xi \vdash C \xrightarrow{a} \Xi \vdash \acute{C}$$

  - Ξ = “context” of C (assumptions + commitments)
  - contains identities + typing of objects and threads known so far
  - checked in incoming communication steps
  - updated in outgoing communication steps

Behavioral interface specification language

Black-box behavior of a component described by a set of traces.

Design goals:

- concise
- fomally justified
- executable in rewriting logic.

$$
\begin{array}{lcll}
\gamma & ::= & x\langle call\ x.I(\vec{x})\rangle \mid x\langle \mathsf{return}(x)\rangle \mid \nu(x{:}T).\gamma \mid (x{:}T).\gamma & \text{basic labels} \\
a & ::= & \gamma? \mid \gamma! & \text{input and output labe} \\
\varphi & ::= & X \mid \epsilon \mid a\ .\ \varphi \mid \varphi + \varphi \mid \mathsf{rec}\ X.\varphi & \text{specifications}
\end{array}
$$

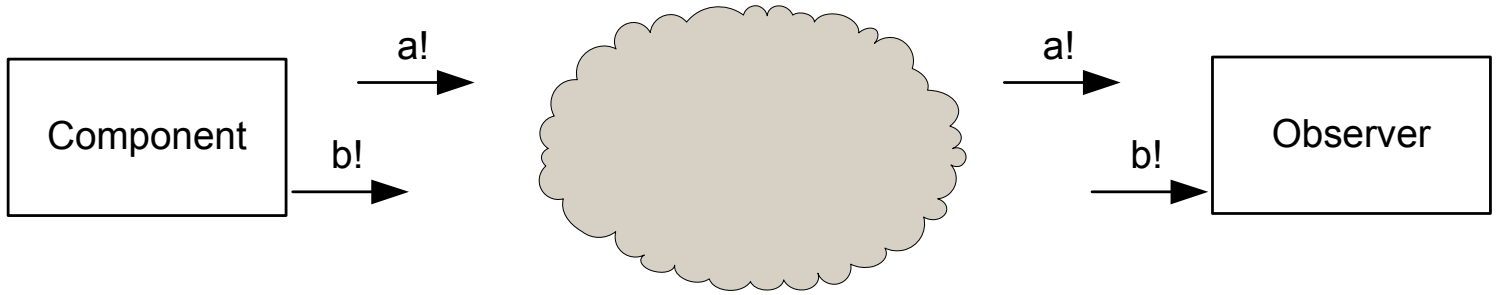
- specification language: uses variables
- two kinds of variable binders
- Creol communication labels: concrete names/references.

Well-formedness

- Restrict specifications to traces actually possible at the interface.
- four three main restrictions:
  - typing
  - scoping
  - communication patterns
  - polarity: specifications either well-formed *input* or well-formed *output*.
- given as *derivation/type system* over trace specs.

Asynchronicity—“Observational blur”

- Asynchronicity: message order not preserved in communication.



- The specification is relaxed up-to observational equivalence
- Testing of output only up-to observability.

$$\frac{}{\nu(\Xi) \,.\, \gamma_1! \,.\, \gamma_2! \,.\, \varphi \equiv_{obs} \nu(\Xi) \,.\, \gamma_2! \,.\, \gamma_1! \,.\, \varphi} \text{ EQ-SWITCH}$$

$$\frac{}{\nu(\Xi) \,.\, \gamma_1! \,.\, \gamma_2! \,.\, \varphi \equiv_{obs} \nu(\Xi) \,.\, \gamma_2! \,.\, \gamma_1! \,.\, \varphi} \text{ EQ-SWITCH}$$

Operational semantics of specifications

Given $\equiv_{obs}$, the meaning of a specification is given operationally and straightforwardly, e.g.:

$$\frac{\acute{\Xi} = \Xi + a}{\Xi \vdash a.\varphi \xrightarrow{a} \acute{\Xi} \vdash \varphi}\ \text{R-PREF} \qquad \frac{\Xi \vdash \varphi_1 \xrightarrow{a} \acute{\Xi} \vdash \varphi_1'}{\Xi \vdash \varphi_1 + \varphi_2 \xrightarrow{a} \acute{\Xi} \vdash \varphi_1'}\ \text{R-PLUS}_1$$

$$\frac{\varphi \equiv_{obs} \varphi' \qquad \Xi \vdash \varphi' \xrightarrow{a} \Xi \vdash \varphi''}{\Xi \vdash \varphi \xrightarrow{a} \Xi \vdash \varphi''}\ \text{R-EQUIV}$$

Asynchronous testing of Creol objects

- Combine:
  - external behavior of object
  - intended behavior given by specification
- interaction defined by synchronous parallel composition
- specification φ and component must engage in corresponding steps:
  - For *incoming* communication, this schedules the order of interactions with the component
  - For *outgoing* communication, the interaction will take place only if it matches an outgoing label in the specification
  - Error if the specification requires input and the component could do output.

Parallel composition

$$\frac{\Xi \vdash C \xrightarrow{\tau} \Xi \vdash \acute{C}}{\Xi \vdash C \parallel \varphi \to \Xi \vdash \acute{C} \parallel \varphi}\ \textsc{Par-Int} \qquad \frac{\vdash a \lesssim_\sigma b \quad \Xi_1 \vdash C \xrightarrow{a} \acute{\Xi}_1 \vdash \acute{C} \qquad \Xi_1 \vdash \varphi \xrightarrow{b} \acute{\Xi}_2 \vdash \acute{\varphi}}{\Xi_1 \vdash C \parallel \varphi \to \acute{\Xi}_1 \vdash \acute{C} \parallel \acute{\varphi}\sigma}\ \textsc{Par}$$

$$\frac{\Xi \vdash \varphi : wf^?}{\Xi \vdash \nu(\Xi').(C \parallel n\langle \mathsf{let}\, x{:}T = o.l(\vec{v})\, \mathsf{in}\, t\rangle \parallel \varphi) \to \lightning}\ \textsc{Err-Call}$$

$$\frac{\Xi \vdash \varphi : wf^?}{\Xi \vdash \nu(\Xi').(C \parallel n\langle v\rangle \parallel \varphi) \to \lightning}\ \textsc{Err-Ret}$$

- Matching of φ's step and components step ($\vdash a \lesssim_\sigma b$)
- As said: specification contains:
  - freshness assertions ($\nu(x{:}T)$)
  - standard variable declarations ($x{:}T$)

Implementation in rewriting logic

- Semantics of Creol is executable in Maude
- Implementation of the spec. language in Maude, too
- Execution of Creol components *synchronized* with specifications
  - *generate* input from specification
  - *test* component behaviour for conformance
- Random generation of input parameters from predefined sets or interval.
- *No input queue*, specified method calls are answered immediately
- Reentering suspended methods may interfere.

Implementation in rewriting logic

- Creol configuration:

```
rl Cfg => Cfg' .
```

- Creol configuration: objects, classes, and messages:

```
rl O C Cfg => O' C M Cfg .
```

- Test framework: introduce `Spec` for specifications.

```
crl Spec || O Cfg => Spec' || O' M Cfg if Cond .
```

- Implementation is close to the operational semantics which is easily coded into Maude.

- “Observational blur”, output prefixes of specifications defined to be AC

Experimental results

- testing by executing parallel composition of component and specification.

```
rew spec || c cClass .
```

- outcomes:
  - error reported
  - stop

- Conformance relation is input-output conformance Execution of c should only lead to output foreseen by `spec`.

- verification by *searching* for error configurations

```
search in PROGRAM :
      spec || c cClass =>+
      spec' || conf errorMsg(S:String)
           such that ....
```

Experiments

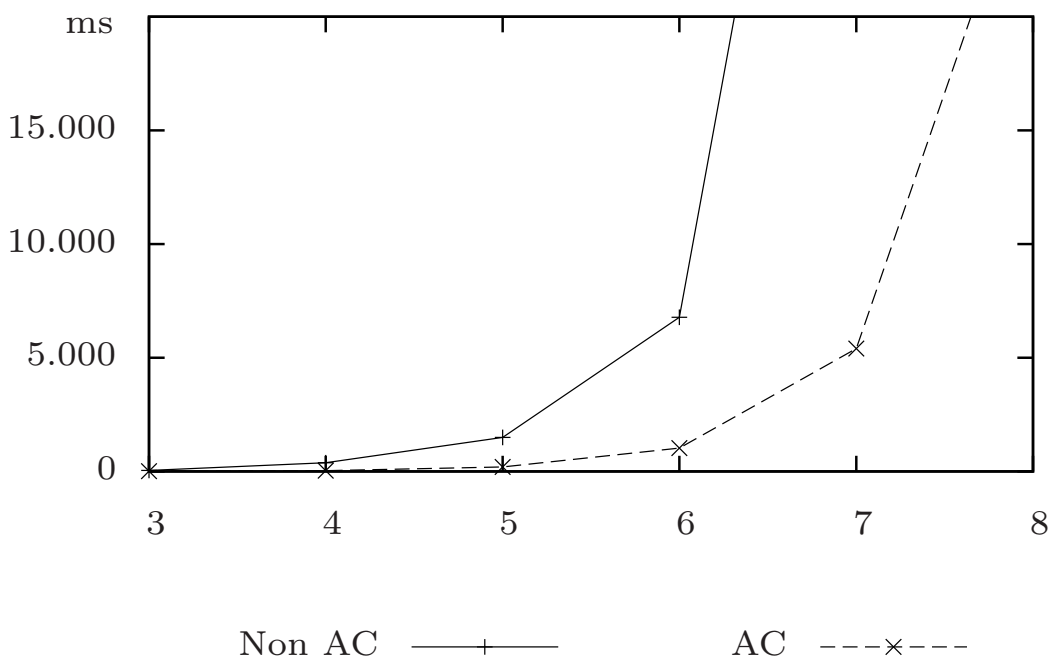
- Experiments to demonstrate usefulness of approach
- Compare rewriting specifications with same semantics but:
  1. using Maude's built in AC rewriting.
  2. equivalent, expanded version of specifications.
- AC rewriting pays off wrt. time and number of rewrites.

Example 1

- Component under test consists of one object with n methods.
- Specification: all methods must have been called before any method may return.
- Tests parametrized over n: spec for $n = 3$:

$$
\begin{aligned}
\mathrm{spec3} \quad = \quad & n_1\langle \mathit{call}\ c.m_1(x_1)\rangle ?\ . \\
& n_2\langle \mathit{call}\ c.m_2(x_2)\rangle ?\ . \\
& n_3\langle \mathit{call}\ c.m_3(x_3)\rangle ?\ . \\
& (n_1\langle \mathit{return}(y_1)\rangle !\ .\ n_2\langle \mathit{return}(y_2)\rangle !\ .\ n_3\langle \mathit{return}(y_3)\rangle !)\ .\ \epsilon
\end{aligned}
$$

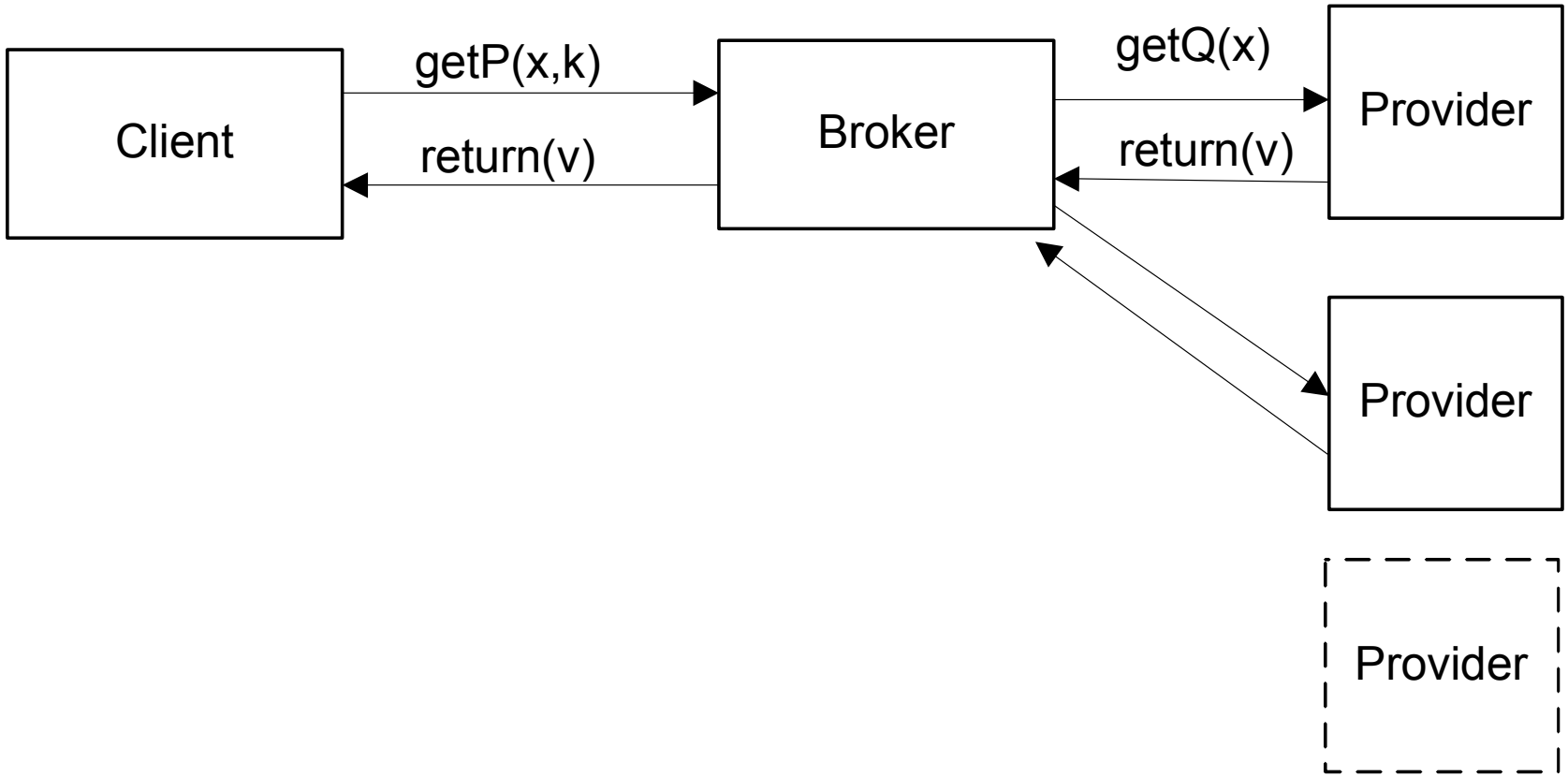
Example 1



n	ms CPU time	
	AC	$Non\ AC$
3	16	47
4	38	379
5	198	1.498
6	1.030	6.782
7	5.407	49.311
8	27.894	NA
9	153.316	NA

Example 2 - broker

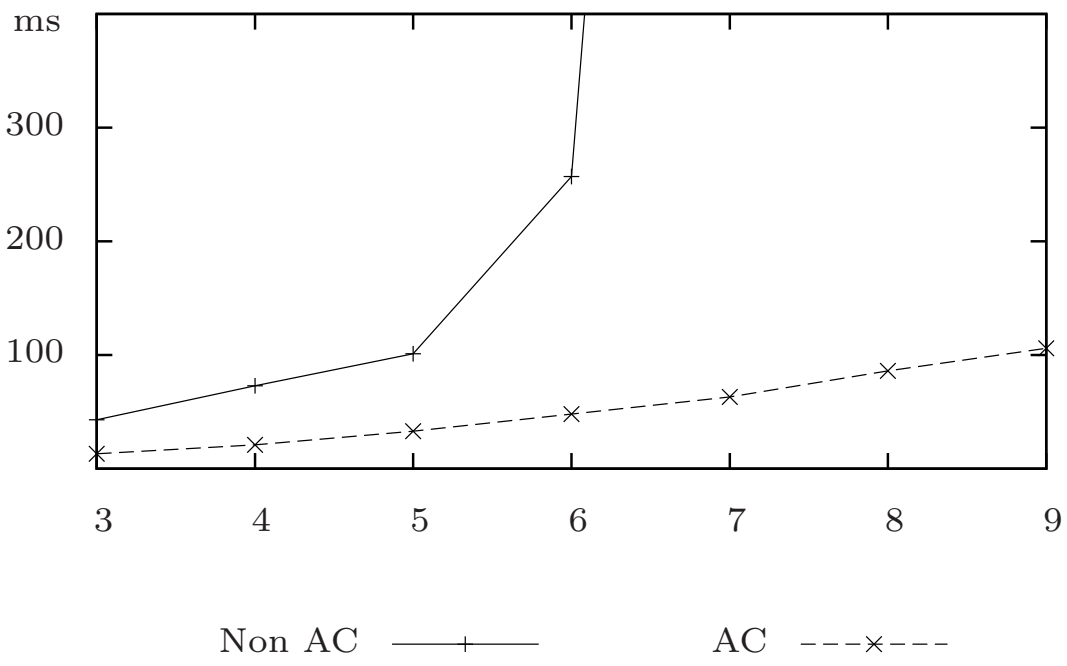
- a broker is an intermediary between client and several providers



- specification: broker must query a certain number of providers before returning

$$\text{specb}_k \quad = \quad n_{c1}\langle \mathit{call}\ b.\mathit{getP}(x,k)\rangle?\ .$$

Example 2



k	ms CPU time	
	AC	$Non\ AC$
3	13	43
4	21	73
5	33	101
6	48	257
7	63	1.965
8	86	17.796
9	106	NA

Summary

- formalization of interface behavior of a concurrent OO language (Creol) + a behavioral interface specification language.
- how to use this specification language for black-box testing of models for asynchronously communicating objects.
- a RW logic formalization of the testing framework for Creol
- using rewriting for conformance testing and search for verification
- one way to deal with potential reordering of communication
- using modulo AC rewriting reduces resource consumption

Future work

- from objects to multi-object components
- extensive case study, testing model for Wireless Sensor Networks.
- extend approach to C# or Java
- narrowing
- use traces from real programs

Related work

- [Tre96] ioco testing
- [VCG+08] observable and controllable actions, conformance based on alternating simulation
- [JOT08] assumption/commitment style verification of components
- [GKST10] formal basis of the approached studied here
- [SAdB+08] testing internal state of Creol objects intra-object scheduling
- [AGSS08] case study for model based testing, using Creol

References I

[AGSS08] Bernhard Aichernig, Andreas Griesmayer, Rudolf Schlatte, and Andries Stam.
Modeling and testing multi-threaded asynchronous systems with Creol.
In *Proceedings of the 2nd International Workshop on Harnessing Theories for Tool Support in Software (TTSS'08)*, ENTCS. Elsevier, 2008.

[GKST10] Immo Grabe, Marcel Kyas, Martin Steffen, and Arild B. Torjusen.
Executable interface specifications for testing asynchronous Creol components.
In Farhad Arbab and Marjan Sirjani, editors, *FSEN*, volume 5961 of *Lecture Notes in Computer Science*, pages 324–339. Springer, 2010.

[JOT08] Einar Broch Johnsen, Olaf Owe, and Arild B. Torjusen.
Validating behavioral component interfaces in rewriting logic.
Fundamenta Informaticae, 82(4):341–359, 2008.

[SAdB+08] Rudolf Schlatte, Bernhard Aichernig, Frank de Boer, Andreas Griesmayer, and Einar Broch Johnsen.
Testing (with) application-specific schedulers for concurrent objects.
2008.
Accepted for ICTAC 2008, 5th International Colloquium on Theoretical Aspects of Computing.

[Tre96] Jan Tretmans.
Test generation with inputs, outputs, and repetetive quiescence.
Software — Concepts and Tools, 17(3):103–120, 1996.

[VCG+08] Margus Veanes, Colin Campbell, Wolfgang Grieskamp, Wolfram Schulte, Nikolai Tillmann, and Lev Nachmanson.
Model-based testing of object-oriented reactive systems with spec explorer.
In *Formal Methods and Testing*, volume 4949 of *Lecture Notes in Computer Science*, pages 39–76. Springer-Verlag, 2008.