Executable interface specifications for testing asynchronous Creol components

I. Grabe, M. Kyas, M. Steffen, and A. B. Torjusen

University of Oslo FU Berlin CWI, Amsterdam

FSEN'09

Background

- Project:
  - asynchronously communicating components in open environments, using Creol
  - behavioral interface description language
  - automated validation techniques
  - testing
- Challenges:
  - asynchronicity
  - non-determinism
- Approach:
  - “divide-and-conquer”
  - black-box behavior given by interactions at the *interface*.

General setting

Goal: Test components under specific schedulings.

Tool: 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 (outline)

1. Theoretical basis:
   - Formalization of the interface behavior of Creol.
   - The behavioral interface specification language.
2. framework for scheduling and asynchronous testing of Creol objects.
3. Implementation of a specification-driven Creol interpreter.

Creol

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

- strongly typed, formal operational semantics in rewriting logic
- features *active objects*.
- communication by *asynchronous* method calls.
- Creol object: acts as a *monitor*.
- *non-deterministic* selection of waiting calls.

Abstract Creol syntax

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

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

Operational semantics

Two stages:

- internal semantics
- external steps occurring at the interface.
- Component/environment: exchange information via *call*- and *return*-labels:

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

- 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* when performing a step

Operational semantics

Two stages:

- internal semantics
- external steps occurring at the interface.
- Component/environment: exchange information via *call*- and *return*-labels:

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

- 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* when performing a step

External steps: outgoing call

$$\frac{a = \nu(\Xi').\ n\langle call\ o.l(\vec{v})\rangle! \qquad \Delta \vdash o \qquad \acute{\Xi} = \Xi + a \quad \Xi' = fn(\lfloor a \rfloor) \cap \Xi_1 \quad \acute{\Xi}_1 = \Xi_1 \setminus \Xi'}{\Xi \vdash \nu(\Xi_1).(C \parallel n\langle \mathsf{let}\, x{:}T = o.l(\vec{v})\ \mathsf{in}\ t\rangle) \xrightarrow{a} \acute{\Xi} \vdash \nu(\acute{\Xi}_1).(C \parallel n'\langle \mathsf{let}\, x : T = n\ \mathsf{in}\, t\rangle)}$$

- label = outgoing call (Δ = assumption context)
- update the contexts
- scope extrusion

External steps: outgoing call

$$\frac{a = \nu(\Xi').\ n\langle call\ o.l(\vec{v})\rangle!\qquad \Delta \vdash o \qquad \acute{\Xi} = \Xi + a \quad \Xi' = fn(\lfloor a \rfloor) \cap \Xi_1 \quad \acute{\Xi}_1 = \Xi_1 \setminus \Xi'}{\Xi \vdash \nu(\Xi_1).(C \parallel n\langle \mathsf{let}\, x{:}T = o.l(\vec{v})\ \mathsf{in}\ t\rangle) \xrightarrow{a} \acute{\Xi} \vdash \nu(\acute{\Xi}_1).(C \parallel n'\langle \mathsf{let}\, x : T = n\ \mathsf{in}\, t\rangle)}$$

- label = outgoing call (Δ = assumption context)
- update the contexts
- scope extrusion

External steps: outgoing call

$$\frac{a = \nu(\Xi').\ n\langle call\ o.l(\vec{v})\rangle!\qquad \Delta \vdash o \qquad \acute{\Xi} = \Xi + a \quad \Xi' = fn(\lfloor a \rfloor) \cap \Xi_1 \quad \acute{\Xi}_1 = \Xi_1 \setminus \Xi'}{\Xi \vdash \nu(\Xi_1).(C \parallel n\langle \mathsf{let}\, x{:}T = o.l(\vec{v})\ \mathsf{in}\ t\rangle) \xrightarrow{a} \acute{\Xi} \vdash \nu(\acute{\Xi}_1).(C \parallel n'\langle \mathsf{let}\, x : T = n\ \mathsf{in}\, t\rangle)}$$

- label = outgoing call (Δ = assumption context)
- update the contexts
- scope extrusion

External steps: outgoing call

$$\frac{a = \nu(\Xi').\ n\langle call\ o.l(\vec{v})\rangle!\qquad \Delta \vdash o \qquad \acute{\Xi} = \Xi + a \quad \Xi' = fn(\lfloor a \rfloor) \cap \Xi_1 \quad \acute{\Xi}_1 = \Xi_1 \setminus \Xi'}{\Xi \vdash \nu(\Xi_1).(C \parallel n\langle \mathsf{let}\, x{:}T = o.l(\vec{v})\ \mathsf{in}\ t\rangle) \xrightarrow{a} \acute{\Xi} \vdash \nu(\acute{\Xi}_1).(C \parallel n'\langle \mathsf{let}\, x : T = n\ \mathsf{in}\, t\rangle)}$$

- label = outgoing call (Δ = assumption context)
- update the contexts
- scope extrusion

External steps

labelled steps at the interface

$$\frac{a = \nu(\Xi').\ n\langle call\ o.l(\vec{v})\rangle?\quad \Xi \vdash a : T\qquad \acute{\Xi} = \Xi + a}{\Xi \vdash C \parallel o[c, F, \bot] \xrightarrow{a} \acute{\Xi} \vdash C \parallel o[c, F, \top] \parallel n\langle \mathsf{let}\, x{:}T = M.l(o)(\vec{v})\,\mathsf{in}\,\mathsf{release}(o); x\rangle}\ \mathsf{CallI}$$

$$\frac{a = \nu(\Xi').\ n\langle call\ o.l(\vec{v})\rangle!\quad \Xi' = fn(\lfloor a \rfloor) \cap \Xi_1\quad \acute{\Xi}_1 = \Xi_1 \setminus \Xi'\quad \Delta \vdash o\quad \acute{\Xi} = \Xi + a}{\Xi \vdash \nu(\Xi_1).(C \parallel n\langle \mathsf{let}\, x{:}T = o.l(\vec{v})\ \mathsf{in}\ t\rangle) \xrightarrow{a} \acute{\Xi} \vdash \nu(\acute{\Xi}_1).(C \parallel n'\langle \mathsf{let}\, x : T = n\ \mathsf{in}\, t\rangle)}\ \mathsf{CallO}$$

$$\frac{a = \nu(\Xi').\ n\langle return(v)\rangle?\qquad \Xi \vdash a : ok\qquad \acute{\Xi} = \Xi + a}{\Xi \vdash C \xrightarrow{a} \acute{\Xi} \vdash C \parallel n\langle v\rangle}\ \mathsf{RetI}$$

$$\frac{a = \nu(\Xi').\ n\langle return(v)\rangle!\qquad \Xi' = fn(\lfloor a \rfloor) \cap \Xi_1\quad \acute{\Xi}_1 = \Xi_1 \setminus \Xi'\quad \acute{\Xi} = \Xi + a}{\Xi \vdash \nu(\Xi_1).(C \parallel n\langle v\rangle) \xrightarrow{a} \acute{\Xi} \vdash \nu(\acute{\Xi}_1).C}\ \mathsf{RetO}$$

Behavioral interface specification language

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

Design goals:

- concise
- intuitive
- executable in rewriting logic

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

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

Behavioral interface specification language

- *distinguish* between input and output interactions:
  - Input: controlled by the environment.
  - Output: to be provided by the component.
- Input interactions are the ones being *scheduled*.
- Output interactions are used for *testing*.

$$\varphi ::= X \mid \epsilon \mid a.\varphi \mid \varphi + \varphi \mid \text{rec } X.\varphi \qquad \text{specifications}$$

- Specially relevant for the choice operator: either external or internal choice.
- Formalized as well-formedness conditions.

Well-formedness

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

Asynchronicity—"Observational blur"

- asynchronicity: messages order not preserved in communication.
- The specification is relaxed up-to *observational equivalence*
- Testing of output only up-to observability.

$$\frac{}{\nu(\Xi).\gamma_1!.\gamma_2!.\varphi \equiv_{obs} \nu(\Xi).\gamma_2!.\gamma_1!.\varphi}\ \textsc{Eq-Switch}$$

Operational semantics of specifications

Given the observational equivalence relation ($\equiv_{obs}$), the meaning of a specification is given operationally in a quite straightforward manner:

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

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

Well-formedness

- $\equiv_{obs}$ preserves well-formedness
- any spec is either in, out or empty
- ϕ is wf! iff ϕ can do an outgoing step (analogously for ?)
- subject reduction: $\Xi \vdash \varphi : wf$ and $\Xi \vdash \varphi \xrightarrow{a} \acute{\Xi} \vdash \acute{\varphi}$, then $\acute{\Xi} \vdash \acute{\varphi} : wf$.
- soundness Assume $\Xi \vdash C$. If $\Xi \vdash C \overset{t}{\Longrightarrow}$, then $\Xi \vdash \varphi_t : wf$ (where φ_t is the trace t interpreted as spec. formula)

Scheduling and 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

$$\frac{\Xi \vdash C \xrightarrow{\tau} \Xi \vdash \acute{C}}{\Xi \vdash C \parallel \varphi \to \Xi \vdash \acute{C} \parallel \varphi}\ \textsc{Par-Int}$$

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

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

- 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 ($x{:}T$)

Matching

$$\frac{}{\vdash () \lesssim () : ok}\ \text{M-Empty} \qquad \frac{\vdash \Xi_1 \lesssim \Xi_2 : ok}{\vdash \nu(n{:}T), \Xi_1 \lesssim \nu(n{:}T), \Xi_2 : ok}\ \text{M-NDec}$$

$$\frac{\vdash \Xi_1 \lesssim \Xi_2 : ok}{\vdash \nu(n{:}T), \Xi_1 \lesssim (n{:}T), \Xi_2 : ok}\ \text{M-Dec}_1 \qquad \frac{\vdash \Xi_1 \lesssim \Xi_2 : ok}{\vdash \Xi_1 \lesssim (n{:}T), \Xi_2 : ok}\ \text{M-Dec}_2$$

$$\frac{\vdash a_1 \lesssim_\sigma a_2 : ok \qquad \vdash \Xi_1 \lesssim \Xi_2\sigma : ok}{\vdash \Xi_1.a_1 \lesssim_\sigma \Xi_2.a_2 : ok}\ \text{M-Lab}$$

Implementation in rewriting logic.

- Creol interpreter 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

- *No input queue*, specified method calls are answered immediately
- Reentering suspended methods may interfere.

Implementation in rewriting logic

- Creol configuration: objects, classes, and messages:
  `rl Cfg => Cfg'.`
- Scheduling interpreter: introduce `Spec` for specifications.
  `rl (Spec || O) Cfg => (Spec' || O') Cfg'.`
- operational semantics easily coded into Maude.
- "Observational blur", implemented rewriting modulo equivalences.

Summary

- Formalization of interface behavior of Creol + a behavioral interface specification language.
- A formal description of how to use this specification language for black-box testing of asynchronously communicating Creol objects.
- A rewriting logic implementation of the testing framework

Future work

- from objects to multi-object components
- more features from the Creol language
- extend specifications with assertion statements on labels
- combine: testing framework + model checking and abstraction
- case study

Related work

- [Johnsen et al., 2008]
  - validating component interfaces
  - assumption/commitment style
  - FOL over traces
- [Schlatte et al., 2008]
  - scheduling activity to restrict behavior
  - intra object scheduling
  - internal state of object

References I

[Johnsen et al., 2008] Johnsen, E. B., Owe, O., and Torjusen, A. B. (2008).
Validating behavioral component interfaces in rewriting logic.
Fundamenta Informaticae, 82(4):341–359.

[Schlatte et al., 2008] Schlatte, R., Aichernig, B., de Boer, F., Griesmayer, A., and Johnsen, E. B. (2008).
Testing (with) application-specific schedulers for concurrent objects.
Accepted for ICTAC 2008, 5th International Colloquium on Theoretical Aspects of Computing.