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.

⇒ 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 (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

\[
C ::= 0 | C \parallel C | \nu(n:T).C \mid n[(O)] \mid n[n, F, L] \mid n\langle t \rangle \\
O ::= F, M \\
M ::= l = m, \ldots, l = m \\
F ::= l = f, \ldots, l = f \\
m ::= ζ(n:T).λ(x:T, \ldots, x:T).t \\
f ::= ζ(n:T).λ().v \mid ζ(n:T).λ().⊥_n \\
t ::= v \mid \text{stop} \mid \text{let } x:T = e \text{ in } t \\
e ::= t \mid \text{if } v = v \text{ then } e \text{ else } e \mid \text{if } \text{undef}(v.l()) \text{ then } e \text{ else } e \\
| \quad v@l(\vec{v}) \mid v.l() \mid v.l := ζ(s:n).λ().v \\
| \quad \text{new } n \mid \text{claim@}(n, n) \mid \text{get@n} \mid \text{suspend}(n) \mid \text{grab}(n) \mid \text{release}(n) \\
v ::= x \mid n \mid () \\
L ::= \perp \mid \top
\]

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

component
object
method suite
fields
method
field
thread
expr.
values
lock status
Operational semantics

Two stages:

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

\[
\begin{align*}
\gamma & ::= n\langle \text{call } n.l(\bar{v}) \rangle \mid n\langle \text{return}(n) \rangle \mid v(n:T).\gamma & \text{basic labels} \\
\alpha & ::= \gamma? \mid \gamma! & \text{input and output labels}
\end{align*}
\]

- External steps

[\Xi \vdash C \xrightarrow{\alpha} \Xi \vdash \hat{C}]

- \Xi = "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:

\[
\gamma ::= n\langle\text{call } n.l(\bar{v})\rangle \mid n\langle\text{return}(n)\rangle \mid \nu(n:T).\gamma \quad \text{basic labels}
\]

\[
a ::= \gamma? \mid \gamma! \quad \text{input and output labels}
\]

- External steps

\[
\Xi \vdash C \xrightarrow{a} \Xi \vdash \hat{C}
\]

- \(\Xi = \text{“context” of } C \) (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

\[
\begin{align*}
a &= v(\Xi'). \ n\langle call \ o.l(\bar{v}) \rangle! & \quad \Delta \vdash o \\
\hat{\Xi} &= \Xi + a & \Xi' = fn([a]) \cap \Xi_1 & \hat{\Xi}_1 = \Xi_1 \setminus \Xi' \\
\Xi \vdash v(\Xi_1). (C \ || \ n\langle \text{let } x : T = o.l(\bar{v}) \text{ in } t \rangle) & \xrightarrow{a} \\
\hat{\Xi} \vdash v(\hat{\Xi}_1). (C \ || \ n'\langle \text{let } x : T = n \text{ in } t \rangle)
\end{align*}
\]

- label = outgoing call (\(\Delta = \) assumption context) 
- update the contexts 
- scope extrusion
External steps: outgoing call

\[ a = \nu(\Xi'). n\langle \text{call } o.l(\vec{v})\rangle! \quad \Delta \vdash o \]

\[
\hat{\Xi} = \Xi + a \quad \Xi' = fn([a]) \cap \Xi_1 \\
\hat{\Xi}_1 = \Xi_1 \setminus \Xi'
\]

\[ \Xi \vdash \nu(\Xi_1). (C \parallel n\langle \text{let } x : T = o.l(\vec{v}) \text{ in } t\rangle) \overset{a}{\rightarrow} \]

\[ \hat{\Xi} \vdash \nu(\hat{\Xi}_1). (C \parallel n'\langle \text{let } x : T = n \text{ in } t\rangle) \]

- label = outgoing call (\(\Delta = \) assumption context)
- update the contexts
- scope extrusion
External steps: outgoing call

\[ a = \nu(\Xi'). \ n\langle\text{call } o.l(\tilde{\nu})\rangle! \quad \Delta \vdash o \]

\[ \tilde{\Xi} = \Xi + a \quad \Xi' = fn([a]) \cap \Xi_1 \quad \tilde{\Xi}_1 = \Xi_1 \setminus \Xi' \]

\[ \Xi \vdash \nu(\Xi_1). (C \parallel n\langle\text{let } x : T = o.l(\tilde{\nu}) \text{ in } t\rangle) \xrightarrow{a} \]

\[ \tilde{\Xi} \vdash \nu(\tilde{\Xi}_1). (C \parallel n'\langle\text{let } x : T = n \text{ in } t\rangle) \]

- label = outgoing call (\(\Delta = \text{assumption context}\))
- update the contexts
- scope extrusion
External steps: outgoing call

\[ a = \nu(\Xi') \quad n\langle \text{call } o.l(\vec{v}) \rangle! \quad \Delta \vdash o \]

\[ \Xi = \Xi + a \quad \Xi' = fn([a]) \cap \Xi_1 \quad \Xi_1 = \Xi_1 \setminus \Xi' \]

\[ \Xi \vdash \nu(\Xi_1).(C \parallel n\langle \text{let } x : T = o.l(\vec{v}) \text{ in } t \rangle) \xrightarrow{a} \]

\[ \Xi \vdash \nu(\Xi_1).(C \parallel n'\langle \text{let } x : T = n \text{ in } t \rangle) \]

- **label** = outgoing call (\(\Delta = \text{assumption context}\))
- update the contexts
- scope extrusion
External steps

labelled steps at the interface

\[ a = \nu(\Xi'). \quad n\langle call \ o.l(\nu)\rangle? \quad \Xi \vdash a : T \quad \dot{\Xi} = \Xi + a \]

\[ \Xi \vdash C || o[c, F, \bot] \xrightarrow{a} \dot{\Xi} \vdash C || o[c, F, \top] || n\langle let x: T = M.l(o)(\nu) in release(o); x\rangle \]

\[ a = \nu(\Xi'). \quad n\langle call \ o.l(\bar{\nu})\rangle! \quad \Xi' = fn([a]) \cap \Xi_1 \quad \dot{\Xi}_1 = \Xi_1 \setminus \Xi' \quad \Delta \vdash o \quad \dot{\Xi} = \Xi + a \]

\[ \Xi \vdash \nu(\Xi_1).(C || n\langle let x: T = o.l(\bar{\nu}) in \ t\rangle) \xrightarrow{a} \dot{\Xi} \vdash \nu(\dot{\Xi}_1).(C || n'\langle let x: T = n in \ t\rangle) \]

\[ a = \nu(\Xi'). \quad n\langle return(\nu)\rangle? \quad \Xi \vdash a : ok \quad \dot{\Xi} = \Xi + a \]

\[ \Xi \vdash C \xrightarrow{a} \dot{\Xi} \vdash C || n\langle \nu\rangle \]

\[ a = \nu(\Xi'). \quad n\langle return(\nu)\rangle! \quad \Xi' = fn([a]) \cap \Xi_1 \quad \dot{\Xi}_1 = \Xi_1 \setminus \Xi' \quad \dot{\Xi} = \Xi + a \]

\[ \Xi \vdash \nu(\Xi_1).(C || n\langle \nu\rangle) \xrightarrow{a} \dot{\Xi} \vdash \nu(\dot{\Xi}_1).C \]
Behavioral interface specification language

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

Design goals:
- concise
- intuitive
- executable in rewriting logic

\[ \gamma ::= x\langle \text{call } x.l(\tilde{x}) \rangle \mid x\langle \text{return}(x) \rangle \mid \nu(x: T).\gamma \mid (x: T).\gamma \]

- basic labels
- input and output specifications

\[ a ::= \gamma? \mid \gamma! \]

\[ \phi ::= X \mid \varepsilon \mid a.\phi \mid \phi + \phi \mid \text{rec } X.\phi \]

- 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*.

\[
\varphi ::= X \mid \varepsilon \mid a.\varphi \mid \varphi + \varphi \mid \text{rec } X.\varphi
\]

- 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.

\[\nu(\Xi).\gamma_1!\gamma_2!\phi \equiv_{obs} \nu(\Xi).\gamma_2!\gamma_1!\phi\]

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

$$\hat{\mathcal{E}} = \mathcal{E} + a$$  \hspace{1cm} \frac{\mathcal{E} \vdash a.\varphi}{\mathcal{E} \vdash \hat{\mathcal{E}} \vdash \varphi} \quad \text{R-PREF}$$

$$\mathcal{E} \vdash \varphi_1 \xrightarrow{a} \hat{\mathcal{E}} \vdash \varphi'_1$$  \hspace{1cm} \frac{\mathcal{E} \vdash \varphi_1 + \varphi_2 \xrightarrow{a} \hat{\mathcal{E}} \vdash \varphi'_1}{\mathcal{E} \vdash \varphi_1 + \varphi_2 \xrightarrow{a} \hat{\mathcal{E}} \vdash \varphi'_1} \quad \text{R-PLUS}_1$$

$$\varphi \equiv_{obs} \varphi'$$  \hspace{1cm} \frac{\mathcal{E} \vdash \varphi' \xrightarrow{a} \mathcal{E} \vdash \varphi''}{\mathcal{E} \vdash \varphi \xrightarrow{a} \mathcal{E} \vdash \varphi''} \quad \text{R-EQUIV}$$
Well-formedness

- $\Xi_{obs}$ preserves well-formedness
- any spec is either in, out or empty
- $\phi$ is wf! iff $\phi$ can do an outgoing step (analogously for ?)
- **subject reduction**: $\Xi \vdash \phi : \text{wf}$ and $\Xi \vdash \phi \xrightarrow{a} \Xi \vdash \phi$, then $\Xi \vdash \phi : \text{wf}$.
- **soundness**: Assume $\Xi \vdash C$. If $\Xi \vdash C \xrightarrow{t}$, then $\Xi \vdash \phi_t : \text{wf}$ (where $\phi_t$ 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 $\phi$ 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

\[\Xi \vdash C \overset{\tau}{\rightarrow} \Xi \vdash \dot{C}\]

\[\Xi \vdash \dot{C} \parallel \varphi \rightarrow \Xi \vdash \dot{C} \parallel \varphi\] \hspace{1cm} \text{PAR-INT}

\[\Xi \vdash \varphi : \text{wf}?\]

\[\Xi \vdash \nu(\Xi').(C \parallel n\langle \text{let } x:T = o.l(\vec{v}) \text{ in } t \rangle \parallel \varphi) \rightarrow \hat{t}\] \hspace{1cm} \text{PAR-ERROR}

\[\Xi_1 \vdash C \overset{a}{\rightarrow} \dot{\Xi}_1 \vdash \dot{C} \quad \Xi_1 \vdash \varphi \overset{b}{\rightarrow} \dot{\Xi}_2 \vdash \varphi \quad \vdash a \lesssim_{\sigma} b\] \hspace{1cm} \text{PAR}

\[\Xi_1 \vdash C \parallel \varphi \rightarrow \dot{\Xi}_1 \vdash \dot{C} \parallel \varphi\sigma\]

- **Matching** of \(\varphi\)'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

\[
\begin{align*}
\vdash () & \preceq () : ok \quad & \text{M-Empty} \\
\vdash \mathbf{\Xi} & \preceq \mathbf{\Xi} : ok \\
\vdash \mathbf{\Xi} & \preceq \mathbf{\Xi} : ok \\
\vdash \mathbf{\Xi} & \preceq \mathbf{\Xi} : ok \\
\end{align*}
\]

\[
\begin{align*}
\vdash (n : T), \mathbf{\Xi} & \preceq (n : T), \mathbf{\Xi} : ok \quad & \text{M-Dec}_1 \\
\vdash (n : T), \mathbf{\Xi} & \preceq (n : T), \mathbf{\Xi} : ok \\
\vdash \mathbf{\Xi} & \preceq (n : T), \mathbf{\Xi} : ok \\
\vdash \mathbf{\Xi} & \preceq (n : T), \mathbf{\Xi} : ok \\
\end{align*}
\]

\[
\begin{align*}
\vdash \sigma a_1 & \preceq \sigma a_2 : ok \\
\vdash \mathbf{\Xi} & \preceq \mathbf{\Xi} \sigma : ok \\
\vdash \mathbf{\Xi} & \preceq \mathbf{\Xi} \sigma : ok \\
\vdash \mathbf{\Xi} & \preceq \mathbf{\Xi} \sigma : ok \\
\end{align*}
\]

\[
\begin{align*}
\vdash \mathbf{\Xi} & \preceq \mathbf{\Xi} \sigma : ok \\
\vdash \mathbf{\Xi} & \preceq \mathbf{\Xi} \sigma : ok \\
\vdash \mathbf{\Xi} & \preceq \mathbf{\Xi} \sigma : ok \\
\vdash \mathbf{\Xi} & \preceq \mathbf{\Xi} \sigma : ok \\
\end{align*}
\]

\[
\begin{align*}
\vdash \Xi_1. a_1 & \preceq \sigma \Xi_2. a_2 : ok \\
\vdash \Xi_1 & \preceq \Xi_2 \sigma : ok \\
\end{align*}
\]

M-NDec
M-Lab
Implementation in rewriting logic.

- Creol interpreter executable in Maude
- Implementation of the spec. language in Maude, too
- Execution of Creol components \textit{synchronized} with specifications
  - \textit{generate} input from specification
  - \textit{test} component behaviour for conformance

- \textit{No input queue}, specified method calls are answered immediately
- Reentering suspended methods may interfere.
Implementation in rewriting logic

- Creol configuration: objects, classes, and messages:
  \[ r1 \text{ Cfg} \Rightarrow \text{Cfg}'. \]
- Scheduling interpreter: introduce Spec for specifications.
  \[ r1 (\text{Spec} \parallel 0) \text{Cfg} \Rightarrow (\text{Spec'} \parallel 0') \text{Cfg}'. \]
- 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