

A Calculus for Ad-Hoc Context Awareness, May 2, 2008

Doina Bucur and Mogens Nielsen

BRICS, Department of Computer Science
University of Aarhus, Denmark
{doina,mn}@brics.dk

1 Syntax

networks	$N ::= a_l[P]$ $N \mid N'$	node at $l, a \in \mathcal{A}, l \in \mathcal{L}$ parallel composition
processes	$P ::= 0$ $P \mid P'$ $P + P'$ $(\nu z)P$ $f(\tilde{z}).P$ $f^\circledast.P$ $\text{def } D$ $\text{def}^\circledast D$ $E P$ $\text{move } l.P$	parallel composition choice restriction, $z \in \mathcal{F}$ local macro call, $f, \tilde{z} \in \mathcal{F}$ reactive macro call local macro definition proactive macro definition public definitions movement
definitions	$E ::= (D)$ $(D)^\circledast$ $D ::= f(\tilde{x}) \triangleright P$ $!f(\tilde{x}) \triangleright P$	locally floating definition floating definition macro, $\tilde{x} \in \mathcal{F}$ permanent macro

Fig. 1. Syntax

A *system* \mathcal{S} consists of:

- a countable set of locations \mathcal{L} ; a location is $l \in \mathcal{L}$
- a countable set of ambients \mathcal{A} ; an ambient is $a \in \mathcal{A}$
- a countable set of macro names \mathcal{F} ; a macro is $f \in \mathcal{F}$
- for each ambient $a \in \mathcal{A}$, two directed graphs: a mobility graph $G_{mob}^a = (\mathcal{L}, E_{mob}^a)$ and a communication graph $G_{com}^a = (\mathcal{L}, E_{com}^a)$.

2 Semantics

$$\begin{array}{c}
\text{MOVE} \quad \frac{(l, l') \in G_{mob}^a}{a_l[\text{move } l'.P \mid Q] \longrightarrow a_{l'}[P \mid Q]} \\
\text{DEF} \quad \text{def } D \longrightarrow (D)0 \quad \text{def}^\ominus D \longrightarrow (D)^\ominus 0 \\
\text{DEFBCAST} \quad \frac{(l, l') \in G_{com}^a}{a_l[(D)^\ominus P] \mid \prod_{b, l'} b_{l'}[Q] \longrightarrow a_l[P] \mid \prod_{b, l'} b_{l'}[(D)Q]} \\
\text{CALL} \quad (f(\tilde{x}) \triangleright Q)f(\tilde{z}).P \longrightarrow \{\tilde{z}/\tilde{x}\}Q \mid P \\
\text{CALLBCAST} \quad \frac{(l, l') \in G_{com}^a}{a_l[f^\ominus \mid P] \mid \prod_{b, l'} b_{l'}[Q] \longrightarrow a_l[P] \mid \prod_{b, l'} b_{l'}[f^\ominus \mid Q]} \\
(f(\tilde{x}) \triangleright Q)f^\ominus \longrightarrow (f(\tilde{x}) \triangleright Q)^\ominus
\end{array}$$

Fig. 2. Operational semantics

The *forwarding capability* ϕ of a network N is a directed multigraph with labeled edges:

$$\phi ::= (\mathcal{L}, E) \quad \text{with} \quad E \subseteq \mathcal{L} \times \mathcal{A} \times \mathcal{L}$$

in which E is a multiset of ordered pairs of vertices with labels from the set of ambient names and $0_\phi = (\mathcal{L}, \emptyset)$.

Two forwarding capabilities are composed simply:

$$(\mathcal{L}, E) \circ (\mathcal{L}, E') = (\mathcal{L}, E \cup E')$$

The forwarding capability of a network is derived inductively:

$$\begin{array}{c}
\text{NULL} \quad 0 : 0_\phi \quad \text{AMB} \quad a_l[P] : (\mathcal{L}, \{(l, a, l') \mid (l, l') \in E_{com}^a\}) \\
\text{NET} \quad \frac{N : \phi \quad N' : \phi'}{N \mid N' : \phi \circ \phi'}
\end{array}$$

Definition 1 (Observability predicates). For any ambient a , location l , definition D , call f and natural number n , the observability predicates are defined by

1. $N \downarrow_{a_l[[\langle D \rangle^{\odot n}]]} \text{ if } N \equiv a_l [[\langle D \rangle^{\odot n}]P] \mid N'$;
2. $N \downarrow_{a_l[\langle D \rangle]} \text{ if } N \equiv a_l [\langle D \rangle P] \mid N'$;
3. $N \downarrow_{a_l[f]} \text{ if } N \equiv a_l [f\langle \tilde{z} \rangle] \mid P \mid N'$;
4. $N \downarrow_{a_l} \text{ if } N \equiv a_l [P] \mid N'$.

3 Expressivity

3.1 Encoding the π Calculus

On the lines of Zimmer's [3], our calculus encodes the monadic, synchronous π calculus with replicated input directly:

$$\begin{aligned}
\llbracket 0 \rrbracket_\pi &\triangleq 0 \\
\llbracket P \mid P' \rrbracket_\pi &\triangleq \llbracket P \rrbracket_\pi \mid \llbracket P' \rrbracket_\pi \\
\llbracket (\nu z)P \rrbracket_\pi &\triangleq (\nu z)\llbracket P \rrbracket_\pi \\
\llbracket \bar{f}\langle z \rangle.P \rrbracket_\pi &\triangleq f\langle z \rangle.\llbracket P \rrbracket_\pi \\
\llbracket f(x).P \rrbracket_\pi &\triangleq \text{def } f(x) \triangleright P \\
\llbracket !f(x).P \rrbracket_\pi &\triangleq \text{def } !f(x) \triangleright P
\end{aligned}$$

In fact, we could have chosen as primitive syntax the asynchronous output $f\langle z \rangle$, without loss of expressivity; this would have encoded the asynchronous version of π (which is as expressive as the fully-synchronous π , by Boudol [1] and Honda and Tokoro [2]).

3.2 Encoding $!P$

Banged processes $!P$ are easily encoded on the basis of banged definitions, $\text{def } !h \triangleright Q$:

$$\llbracket !P \rrbracket \triangleq (\nu h) (\text{def } !h \triangleright P \mid h)$$

3.3 Encoding Multihop Broadcasts

Based on the semantics for broadcasting a definition over a single hop, we encode multihop broadcasts, written $(D)^{\odot n}$, $n \in \mathbb{N}$. We assume a unique macro name fwd , known and permanently called (there exists an encoding for banged calls, as in Subsection 3.2) by all agents in the network; all agents' running of $!fwd$ means to say that all agents agree to be forwarders of definitions (and get a copy of any forwarded definition). The encoding is as follows:

$$\begin{aligned}
\llbracket (D)^{\odot 1} \rrbracket &\triangleq (D)^{\odot} \\
\llbracket (D)^{\odot n} \rrbracket &\triangleq (fwd \triangleright (D)(D)^{\odot n-1} 0)^{\odot} \text{ for } n > 1
\end{aligned}$$

For example, have a 2-hop definition broadcast, with $(l, l') \in G_{com}^a$:

$$\begin{aligned}
&a_l[\llbracket (D)^{\odot 2} \rrbracket P] \mid b_{l'}[!fwd \mid Q] \\
&\triangleq a_l[(fwd \triangleright (D)(D)^{\odot 0})^{\odot} P] \mid b_{l'}[!fwd \mid Q] \\
&\longrightarrow a_l[P] \mid b_{l'}[(fwd \triangleright (D)(D)^{\odot 0})(!fwd \mid Q)] \\
&\equiv \longrightarrow a_l[P] \mid b_{l'}[(D)(D)^{\odot 0} !fwd \mid Q] \\
&\equiv a_l[P] \mid b_{l'}[(D)(D)^{\odot} !fwd \mid Q]
\end{aligned}$$

from which b proceeds with the singlehop broadcast $(D)^{\odot}$ as usual. We can prove that the encoding is correct:

Lemma 1 (Multihop definition broadcasts). *Have $N : \phi$ immobile with $N \Downarrow_{a_l}[\llbracket (D)^{\odot n} \rrbracket]$. If $a\text{-dist}_\phi(l, l') = n$, then if $N \Downarrow_{b_{l'}}$ it can hold that $N \Downarrow_{b_{l'}}[(D)]$.*

In other words, in an immobile setting, a destination n edges away (in the forwarding capability ϕ) from a source can be reached with a n -hop transmission. Also, a multihop transmission behaves like a singlehop transmission generated by an ambient with a communication graph having direct edges from the source to the destinations n hops away.

4 Case Study: a Distributed Coordination Protocol for Monitoring Spatial Phenomena

Have a finite number of locations, each with an immobile sensor offering fresh readings at all times. A smaller number of mobile robots need to gather as updated as possible sensor readings for all locations; they do this by either visiting the sensor's location themselves, or by exchanging readings with robots at the same location.

Each robot having arrived at a new location will:

1. get an updated reading from the local sensor;
2. if another robot is present, adopt the other robot's readings when they are newer than its own; also, provide its own newer readings to the other robot;
3. using the resulting updated database, use heuristics to compute next move; execute move;
4. download the sensor readings upon request.

New syntax and notations:

- a comparison between two sensor readings, age-wise: $newer(d, d')$ returns a boolean true if the macro call d points to an newer reading than that of d' ;
- an if construct: $if\ bool\ P$ runs process P if $bool$ evaluates to true;
- a compact notation and semantics (derived from the standard semantics) for requesting a remote, permanent definition instead of a single copy; $!f^\ominus$ provokes a permanent definition to be broadcast from the neighbour, if a permanent definition is indeed available at the neighbour:

$$\text{!CALLBCAST} \frac{(l, l') \in G_{com}^a}{a_l[!f^\ominus \mid P] \mid \prod_{b, l'} b_{l'}[Q] \longrightarrow a_l[P] \mid \prod_{b, l'} b_{l'}[!f^\ominus \mid Q]}$$

$$(!f(\tilde{x}) \triangleright Q)!f^\ominus \longrightarrow (!f(\tilde{x}) \triangleright Q)^\ominus$$

4.1 A Sensor

A sensor reading is a process P (think of it as a database entry) pointed to by a macro name, e.g. $d \triangleright P$. We abstract away from the implementation of the sensor reading; we only assume a boolean age comparison between two such macros pointing to sensor readings $newer(reading1, reading1')$.

A sensor senses its new reading P by calling a well-known macro name $sense$ and passing it a new name in which to store the reading, d :

$$(sense(y) \triangleright def\ !y \triangleright P) \nu d\ sense\langle d \rangle$$

Also, sensor k broadcasts its reading to any robot by defining a well-known name $sreading_k$ to provide the name of that newest reading d , followed by an instruction for the receiving robot to continue executing its internal loop l with

the new data from this sensor d and the robot's own existing data from the other sensors \tilde{e} :

$$def^\circ sreading_k(l, \tilde{e}) \triangleright !d^\circ \mid l\langle d, \tilde{e} \rangle$$

The sensor's general behaviour is a recursive loop of either sensing a new reading or broadcasting its current reading, as described above; the loop's parameter d is passed the name of the current reading, and the loop only starts after a first reading:

$$\begin{aligned} & \nu loop \\ & (!loop(d) \triangleright \\ & \quad \nu d \text{ sense}\langle d \rangle . loop\langle d \rangle \\ & \quad + def^\circ (sreading_k(l, \tilde{e}) \triangleright !d^\circ . l\langle d, \tilde{e} \rangle) \mid loop\langle d \rangle) \\ & (\text{sense}(y) \triangleright def \ !y \triangleright P) \\ & \nu d \text{ sense}\langle d \rangle . loop\langle d \rangle \end{aligned}$$

4.2 A Robot

A robot updates one of its sensor readings from nearby sensor k , passing it the names of its loop $loop$ and of its existing readings (minus that of sensor k) \tilde{e} :

$$sreading_k\langle loop, \tilde{e} \rangle$$

If another robot is nearby, our robot broadcasts to the neighbour its newer readings (in the fashion of a sensor broadcasting its single reading) over well-known names $rreading_k$, complete with an instruction to loop:

$$if \text{ newer}(\dots) def^\circ rreading_k(l, \tilde{e}) \triangleright !e_k^\circ . l\langle e_k, \tilde{e} \rangle$$

It can also be at the other end of the above broadcast:

$$rreading_k\langle loop, \tilde{e} \rangle$$

or it can download a copy of the readings to whomever calls *download*:

$$(download \triangleright \prod_k e_k^\circ)^\circ$$

Altogether, a robot executes a recursive loop of either updating its readings (from any sensor k or robot), or sending them out (to another robot), or downloading a copy of the readings. The loop is passed as parameters the list of the names of all current readings \tilde{e} , in which e_k is the current reading from sensor k . The list is initially \widetilde{NA} :

$$\begin{aligned} & \nu loop \\ & (!loop(\tilde{e}) \triangleright \\ & \quad (download \triangleright \prod_k e_k^\circ)^\circ \\ & \quad \sum_k sreading_k\langle loop, \tilde{e} \rangle \\ & \quad + \sum_k rreading_k\langle loop, \tilde{e} \rangle \\ & \quad + \sum_k if \text{ newer}(\dots) def^\circ rreading_k(l, \tilde{e}) \triangleright !d^\circ . l\langle d, \tilde{e} \rangle \mid loop\langle \tilde{e} \rangle) \\ & loop\langle \widetilde{NA} \rangle \end{aligned}$$

One could use some examples of how the protocol unfolds here.

5 Case Study: a Cooperative Protocol for Tracking

From "Pervasive Pheromone-Based Interaction with RFID Tags", M. Mamei and F. Zambonelli, ACM Transactions on Autonomous and Adaptive Systems, Vol. 2, No. 2, Article 4, Publication date: June 2007. Pages 1-13.

References

1. Gerard Boudol. Asynchrony and the pi-calculus. Technical Report RR-1702, INRIA.
2. Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. *Lecture Notes in Computer Science*, 512:133–147, 1991.
3. Pascal Zimmer. A calculus for context-awareness. Technical Report BRICS Report Series RS-05-27.