SHReQ: A Framework for Coordinating Application Level QoS

Dan Hirsch
dhirsch@di.unipi.it

http://www.di.unipi.it/~dhirsch

Emilio Tuosto
etuosto@di.unipi.it

http://www.di.unipi.it/~etuosto

Dipartimento di Informatica, Università di Pisa

QAPL: Edinburgh, 2 – 3 April 2005
Overview of the talk

“Ayudadme a comprender lo que os digo y os lo explicaré mejor”
“Help me in understanding what I’m saying and I will explain it better”

(Antonio Machado)
Overview

- A few motivations
- Background
  - application level QoS
  - constraint semirings (c-semirings)
  - Synchronised Hyperedge Replacement (SHR)
- Putting things together: SHReQ
  - weighted graphs...
  - ...and their productions
  - synchronising productions for SHReQ
- An example
- Conclusions
Motivations

The real technology - behind all of our technologies - is language

(N. Fisher)
Global Computing

Programming *global systems* is hard because:

- Absence of centralised control (*self*)
- Client-Server not enough: P2P
- Administrative domains (*Security*)
- Interoperability
  - different platforms
  - different devices (e.g. PDA, laptop, mobile phones...)
- “Mobility” (resources & computation)
- Network Awareness
  - Applications are location dependent
  - Locations have different features
  - and allow multiple access policies
- Independently programmed in a distributed environment
- ...
Service Oriented Computing

- applications are made by gluing services
  - “autonomous”
  - independent (local choices, independently built)
  - mobile/stationary
  - “interconnected”
- interactions governed by programmable coordination policies
- services are searched and binded ... offline
Service Oriented Computing

- applications are made by gluing services
  - “autonomous”
  - independent (local choices, independently built)
  - mobile/stationary
  - “interconnected”
- interactions governed by programmable coordination policies
- services are searched and binded ... offline

Can search/bind be dynamic and at run-time?
Global Computing and Services

Service Oriented Computing

- applications are made by gluing services
  - “autonomous”
  - independent (local choices, independently built)
  - mobile/stationary
  - “interconnected”
- interactions governed by programmable coordination policies
- services are searched and binded ... offline

Can search/bind be dynamic and at run-time?

Search and bind wrt application level QoS

- not low-level performance (e.g., throughput, response time)
- but application-related, e.g.
  - price services
  - payment mode
  - data available in a given format
Our approach in brief

WAN programming is not just $\text{go}(P)$, $\bar{s}(x)$ or $s(y)$

- Lifting QoS issues to application level...
- ...for programming global computers
- with programmable application level QoS
- and develop proof techniques and tools

We are currently defining **SHReQ**, an (hyper)graph model which exploits **c-semiring** for
- expressing application level QoS and
- for coordinating activities...
- ...by synchronisation on c-semiring values

First steps (extending **Klaim**) in [DFM+03, DFM+05]
Background

“During my nine years at the elementary schools
I was not able to teach anything to my professors”

(Bertolt Brecht)
### Constraint Semirings

C-Semirings $[\text{BMR95, BMR97}]$ for abstracting application level QoS

- $\langle A, +, \ast, 0, 1 \rangle$, where
  - $A$ is a set (containing 0 and 1),
  - $+, \ast : A \times A \rightarrow A$

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x + y = y + x$</td>
<td>$x \ast y = y \ast x$</td>
</tr>
<tr>
<td>$(x + y) + z = x + (y + z)$</td>
<td>$(x \ast y) \ast z = x \ast (y \ast z)$</td>
</tr>
<tr>
<td>$x + 0 = x$</td>
<td>$x \ast 0 = 0$</td>
</tr>
<tr>
<td>$x + 1 = 1$</td>
<td>$x \ast 1 = x$</td>
</tr>
<tr>
<td>$x + x = x$</td>
<td>$(x + y) \ast z = (x \ast z) + (y \ast z)$</td>
</tr>
</tbody>
</table>
## Constraint Semirings

### C-Semirings \([\text{BMR95, BMR97}]\) for abstracting application level QoS

- \(\langle A, +, *, 0, 1 \rangle\), where
  - \(A\) is a set (containing 0 and 1),
  - \(+, * : A \times A \rightarrow A\)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>(x + y = y + x)</td>
</tr>
<tr>
<td></td>
<td>((x + y) + z = x + (y + z))</td>
</tr>
<tr>
<td></td>
<td>(x + 0 = x)</td>
</tr>
<tr>
<td></td>
<td>(x + 1 = 1)</td>
</tr>
<tr>
<td></td>
<td>(x + x = x)</td>
</tr>
<tr>
<td>(\ast)</td>
<td>(x \ast y = y \ast x)</td>
</tr>
<tr>
<td></td>
<td>((x \ast y) \ast z = x \ast (y \ast z))</td>
</tr>
<tr>
<td></td>
<td>(x \ast 0 = 0)</td>
</tr>
<tr>
<td></td>
<td>(x \ast 1 = x)</td>
</tr>
<tr>
<td></td>
<td>((x + y) \ast z = (x \ast z) + (y \ast z))</td>
</tr>
</tbody>
</table>
Constraint Semirings

C-Semirings \([\text{BMR95, BMR97}]\) for abstracting application level QoS

- \(\langle A, +, *, 0, 1 \rangle\), where
- \(A\) is a set (containing 0 and 1),
- \(+, * : A \times A \rightarrow A\)

<table>
<thead>
<tr>
<th>+</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x + y = y + x)</td>
<td>(x * y = y * x)</td>
</tr>
<tr>
<td>((x + y) + z = x + (y + z))</td>
<td>((x * y) * z = x * (y * z))</td>
</tr>
<tr>
<td>(x + 0 = x)</td>
<td>(x * 0 = 0)</td>
</tr>
<tr>
<td>(x + 1 = 1)</td>
<td>(x * 1 = x)</td>
</tr>
<tr>
<td>(x + x = x)</td>
<td>((x + y) * z = (x * z) + (y * z))</td>
</tr>
</tbody>
</table>

- Implicit partial order: \(a \leq b \iff a + b = b\) “\(b\) is better than \(a\)”
Examples of c-semirings

Example 1 (Priority) \( P = \langle N, \max, \min, 0, \infty \rangle \) where \( N \) is the set of natural numbers with infinity.

Example 2 (Broadcast) \( B = \langle A \cup \overline{A} \cup \{1, 0, \perp\}, 0, 1, +, \star \rangle \) where \( A \) is a set of actions, \( \overline{A} = \{\overline{a} | a \in A\} \) are the coactions and

\[
\forall a \in \text{Act}. a \star \overline{a} = \overline{a} \land a \star a = a
\]

\[
\forall a, b \in \text{Act} \cup \overline{\text{Act}} \cup \{\perp\} : b \notin \{a, \overline{a}\} \implies a \star b = \perp
\]

the corresponding commutative rules plus the ones for 0 and 1 + also obeys the axioms

\[
\begin{align*}
a + a &= a \\
a + b &= \perp
\end{align*}
\]

\[\forall a, b \in \text{Act} \cup \overline{\text{Act}} \cup \{\perp\}. b \neq a\]

Proposition 1 Cartesian product of c-semirings is a c-semiring.

e.g., \( BP = B \times P \) is the c-semiring of broadcast with priorities
Another bunch of c-semiring examples

C-semirings structures can be defined for many frameworks:

- \( \langle \{ \text{true, false} \}, \lor, \land, \text{false, true} \rangle \) (boolean): Availability
- \( \langle \text{Real}^+, \min, +, +\infty, 0 \rangle \) (optimization): Price, propagation delay
- \( \langle \text{Real}^+, \max, \min, 0, +\infty \rangle \) (max/min): Bandwidth
- \( \langle [0, 1], \max, \cdot, 0, 1 \rangle \) (probabilistic): Performance and rates
- \( \langle [0, 1], \max, \min, 0, 1 \rangle \) (fuzzy): Performance and rates
- \( \langle 2^N, \cup, \cap, \emptyset, N \rangle \) (set-based, where \( N \) is a set): Capabilities and access rights
Hypergraphs model

- Distributed systems as graphs [CM83, DM87]
  - explicitly describe topology
  - and are suitable for expressing multiparty synchronisation
  - we use Synchronised Hyperedge Replacement (SHR)

- Edge replacement for graph rewritings [Fed71, Pav72]
- Edge replacement/distributed constraint solving problem [MR96]
- Graphs grammars for software architecture styles [HIM00]
- SHR with mobility for nominal calculi [HM01, Hir03]
- Extension to node fusions [FMT01, Tuo03]...
- ...which accounts for a concurrent semantics of the Fusion calculus [LM03]
Hypergraphs model

We aim at tackling new *non-functional* computational phenomena of systems using SHR.

The metaphor is

- “Global computers *as* Hypergraphs”
- “Global computing *as* SHR”

In other words:

- Components are represented by hyperedges
- Systems are *bunches* of (connected) hyperedges
- Computing means to synchronously rewrite hyperedges...
- ...according to a synchronisation policy
\[ L \rightarrow G \]
$\text{SHR...naively}$

$L \rightarrow G$

$$G_1 \quad 1 \quad 2 \quad L \quad 4 \quad 5 \quad G_2$$

Beneﬁts:
- Uniform framework for
- LTS for Ambient
- Fusion
- ... for
- ... and
- path reservation
- ... for distributed coordination
- wireless networks

[Tuo05]
$L \rightarrow G$
$L \rightarrow G$

- Edge replacement: local
- Synchronisation as distributed constraint solving
- Multiple synchronisation
- New node creation
- Node fusion: model of mobility and communication
- Truly concurrent semantics
L → G

- Edge replacement: local
- Synchronisation as distributed constraint solving
- Multiple synchronisation
- New node creation
- Node fusion: model of mobility and communication
- Truly concurrent semantics

Benefits:
- Uniform framework for π, π-!, Fusion
- LTS for Ambient ...
- ... for Klaim ...
$L \rightarrow G$

- Edge replacement: local
- Synchronisation as distributed constraint solving
- Multiple synchronisation
- New node creation
- Node fusion: model of mobility and communication
- Truly concurrent semantics

Benefits:
- Uniform framework for $\pi$, $\pi$-I, Fusion
- LTS for Ambient ...
- ... for Klaim ...

- ... and path reservation for a Klaim extension
- expressive for distributed coordination
- wireless networks [Tuo05]
“[...] Shrek cuts a deal with Farquaad and sets out to rescue the beautiful Princess Fiona to be Farquaad’s bride.”

(from http://www.shrek.com)
Hyperedges and Hypergraphs Syntax

A hyperedge connects more than two nodes (generalisation of edge)

\[ L : 3, \ L(y, z, x), \]

\[ x \rightarrow 3 \rightarrow L \rightarrow 2 \rightarrow z \]
A hyperedge connects more than two nodes (generalisation of edge)

\[ L : 3, \quad L(y, z, x), \]

\[ y \]

\[ \llap{x} \bullet \overrightarrow{3} \overset{}{L} \overrightarrow{2} \bullet z \]

\[ G ::= \text{nil} \mid L(x) \mid G|G \mid \nu \ y . G \]
Hyperedges and Hypergraphs Syntax

A hyperedge connects more than two nodes (generalisation of edge)

\[ L : 3, \quad L(y, z, x), \]

\[
\begin{array}{c}
\text{Syntactic Judgement} \\
\quad x_1 : s_1, \ldots, x_n : s_n \vdash G, \quad fn(G) \subseteq \{x_1, \ldots, x_n\}
\end{array}
\]
Hyperedges and Hypergraphs Syntax

A hyperedge connects more than two nodes (generalisation of edge)

\[ L : 3, \ L(y, z, x), \]

\[
\begin{array}{c}
\bullet \\
\uparrow \quad 1 \\
\bullet \\
\end{array}
\]

\[
x \bullet \rightarrow^3 \quad L \rightarrow^2 \bullet z
\]

\[ G ::= \text{nil} \mid L(x) \mid G|G \mid \nu \ y.\ G \]

Syntactic Judgement

\[ x_1 : s_1, \ldots, x_n : s_n \vdash G, \quad fn(G) \subseteq \{x_1, \ldots, x_n\} \]

An example:

\[ L : 3, \quad M : 2 \]

\[ x : 1, \ y : 0 \vdash \nu \ z.(L(y, z, x)|M(y, z)) \]
Hyperedges and Hypergraphs Syntax

A hyperedge connects more than two nodes (generalisation of edge)

\[ L : 3, \ L(y, z, x), \]

\[ G ::= \text{nil} \mid L(x) \mid G \mid \nu y. G \]

Syntactic Judgement

\[ x_1 : s_1, \ldots, x_n : s_n \vdash G, \quad fn(G) \subseteq \{x_1, \ldots, x_n\} \]

An example:

\[ L : 3, \quad M : 2 \]

\[ x : 1, \ y : 0 \vdash \nu z. (L(y, z, x) | M(y, z)) \]
Productions of SHReQ are based on requirements: $\mathcal{R} = S \times \mathcal{N}^*$ (where $\langle S, +, \cdot, 0, 1 \rangle$ is a fixed c-semiring)

production $\xi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G$
Productions of SHReQ are based on requirements: $\mathcal{R} = S \times \mathcal{N}^*$ (where $\langle S, +, \cdot, 0, 1 \rangle$ is a fixed c-semiring)

Production $\chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G$

$\tilde{x}$ is a tuple of pairwise distinguished nodes and $L : |\tilde{x}|$
Productions of **SHReQ** are based on **requirements**: \( \mathcal{R} = S \times \mathcal{N}^* \) (where \( \langle S, +, \ast, 0, 1 \rangle \) is a fixed c-semiring)

- \( \tilde{x} \) is a tuple of pairwise distinguished nodes and \( L : |\tilde{x}| \)
- \( \chi : \{|\tilde{x}|\} \rightarrow S \) is the **applicability function**
Productions of SHReQ are based on requirements: \( R = S \times \mathcal{N}^* \) (where \( \langle S, +, \times, 0, 1 \rangle \) is a fixed c-semiring)

\[
\chi \triangleright L(\vec{x}) \xrightarrow{\Lambda} G
\]

- \( \vec{x} \) is a tuple of pairwise distinguished nodes and \( L : |\vec{x}| \)
- \( \chi : \{|\vec{x}|\} \to S \) is the applicability function
- \( \Lambda : \{|\vec{x}|\} \to R \) is the communication function.

\( n(\Lambda) \) communicated nodes of \( \Lambda \): those nodes appearing in a requirement in the range of \( \Lambda \).

The set of new nodes of \( \Lambda \) is \( \text{new}(\Lambda) = n(\Lambda) \setminus \text{dom}(\Lambda) \)
Productions of **SHReQ** are based on requirements: \( \mathcal{R} = S \times \mathcal{N}^* \)

(where \( \langle S, +, *, 0, 1 \rangle \) is a fixed c-semiring)

- \( \tilde{x} \) is a tuple of pairwise distinguished nodes and \( L : |\tilde{x}| \)
- \( \chi : \{ |\tilde{x}| \} \to S \) is the applicability function
- \( \Lambda : \{ |\tilde{x}| \} \to \mathcal{R} \) is the communication function.
  - \( n(\Lambda) \) communicated nodes of \( \Lambda \): those nodes appearing in a requirement in the range of \( \Lambda \)
  - The set of new nodes of \( \Lambda \) is \( \text{new}(\Lambda) = n(\Lambda) \setminus \text{dom}(\Lambda) \)
- \( G \) is a graph s.t. \( \text{fn}(G) \subseteq \{ |\tilde{x}| \} \cup n(\Lambda) \)
Interpreting SHReQ productions

Consider

\[ \pi : \chi \triangleright L(\tilde{x}) \xRightarrow{\Lambda} G \]

and a graph \( H \) having an arc labelled by \( L \), e.g.:
Interpreting \textbf{SHReQ} productions

Consider

\[ \pi : \chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G \]

and a graph \( H \) having an arc labelled by \( L \), e.g.:

Replacing \( L \) with \( G \) in \( H \) according to \( \pi \) requires that \( H \) satisfies the conditions expressed by \( \chi \) on the attachment nodes of \( L \).
Interpreting SHReQ productions

Consider

\[ \pi : \chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G \]

and a graph \( H \) having an arc labelled by \( L \), e.g.:

Replacing \( L \) with \( G \) in \( H \) according to \( \pi \) requires that \( H \) satisfies the conditions expressed by \( \chi \) on the attachment nodes of \( L \).
Interpreting \textit{SHReQ} productions

Consider

\[ \pi : \chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G \]

and a graph \( H \) having an arc labelled by \( L \), e.g.:

Replacing \( L \) with \( G \) in \( H \) according to \( \pi \) requires that \( H \) satisfies the conditions expressed by \( \chi \) on the attachment nodes of \( L \).

Once \( \chi \) is satisfied in \( H \), \( L(\tilde{x}) \) contributes to the rewriting by offering \( \Lambda \) in the synchronisation with all the edges connected to nodes in \( \tilde{x} \).
Interpreting SHReQ productions

Consider

\[ \pi : \chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G \]

and a graph \( H \) having an arc labelled by \( L \), e.g.:

Replacing \( L \) with \( G \) in \( H \) according to \( \pi \) requires that \( H \) satisfies the conditions expressed by \( \chi \) on the attachment nodes of \( L \).

Once \( \chi \) is satisfied in \( H \), \( L(\tilde{x}) \) contributes to the rewriting by offering \( \Lambda \) in the synchronisation with all the edges connected to nodes in \( \tilde{x} \).
Synchronised Rewriting for **SHReQ**

**Events for**

<table>
<thead>
<tr>
<th><strong>Synchronisation</strong></th>
<th><strong>Sync</strong> and <strong>Fin</strong> s.t.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync ⊆ Fin ⊆ S</td>
<td></td>
</tr>
<tr>
<td>1 ∈ Sync</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>No synchronisation</strong></th>
<th><strong>NoSync</strong> ⊆ S \ Fin s.t.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S ⋈ NoSync ⊆ NoSync</td>
<td></td>
</tr>
<tr>
<td>0 ∈ NoSync</td>
<td></td>
</tr>
</tbody>
</table>

**SHReQ semantics** exploits a mgu accounting for node fusions.

Let \( \Omega \) be a finite multiset over \( \mathcal{N} \times \mathcal{R} \): \( \text{mgu}(\Omega) \) for denoting an idempotent substitution is defined iff

\[
(x, s, \tilde{u}), (x, s', \tilde{v}) \in \Omega @ x \\
\text{implies} \quad |	ilde{u}| = |	ilde{v}| \\
\forall i \in \{1, \ldots, |\tilde{u}|\} : \tilde{u}_i \in \text{new}(\Omega) \lor \tilde{v}_i \in \text{new}(\Omega) \\
|\Omega @ x| > 1 \implies \prod_{(x, s, \tilde{y}) \in \Omega @ x} s \notin \text{NoSync}
\]
Synchronised Rewriting for SHReQ

Events for

**Synchronisation** $\text{Sync}$ and $\text{Fin}$ s.t.
- $\text{Sync} \subseteq \text{Fin} \subseteq S$
- $1 \in \text{Sync}$

**No synchronisation** $\text{NoSync} \subseteq S \setminus \text{Fin}$ s.t.
- $S \ast \text{NoSync} \subseteq \text{NoSync}$
- $0 \in \text{NoSync}$

SHReQ semantics exploits a mgu accounting for node fusions.

Let $\Omega$ be a finite multiset over $\mathcal{N} \times \mathcal{R}$: $\text{mgu}(\Omega)$ for denoting an idempotent substitution is defined iff

\[
(x, s, \tilde{u}), (x, s', \tilde{v}) \in \Omega @ x
\]
implies

\[
|\tilde{u}| = |\tilde{v}|
\]

\[
\forall i \in \{1, \ldots, |\tilde{u}|\} : \tilde{u}_i \in \text{new}(\Omega) \lor \tilde{v}_i \in \text{new}(\Omega)
\]

\[
|\Omega @ x| > 1 \implies \prod_{(x, s, \tilde{y}) \in \Omega @ x} s \notin \text{NoSync}
\]
Synchronised Rewriting for SHReQ

Events for

**Synchronisation** $\text{Sync}$ and $\text{Fin}$ s.t.
- $\text{Sync} \subseteq \text{Fin} \subseteq S$
- $1 \in \text{Sync}$

**No synchronisation** $\text{NoSync} \subseteq S \setminus \text{Fin}$ s.t.
- $S \cap \text{NoSync} \subseteq \text{NoSync}$
- $0 \in \text{NoSync}$

SHReQ semantics exploits a mgu accounting for node fusions.

Let $\Omega$ be a finite multiset over $\mathcal{N} \times \mathcal{R}$: $\text{mgu}(\Omega)$ for denoting an idempotent substitution is defined iff

\[(x, s, \tilde{u}), (x, s', \tilde{v}) \in \Omega \at x\]

implies

\[|\tilde{u}| = |\tilde{v}|\]

\[\forall i \in \{1, \ldots, |\tilde{u}|\} : \tilde{u}_i \in \text{new}(\Omega) \lor \tilde{v}_i \in \text{new}(\Omega)\]

\[|\Omega \at x| > 1 \implies \prod_{(x, s, \tilde{y}) \in \Omega \at x} s \notin \text{NoSync}\]

and obtained by computing the mgu of the equations

\[\{\tilde{u}_i = \tilde{v}_i | \exists s, t \in S : (x, s, \tilde{u}), (x, t, \tilde{v}) \in \Omega \land 1 \leq i \leq |\tilde{u}|\}\]
Quasi-productions

The set $QP$ of quasi-productions on $P$ is the smallest set s.t. $P \subseteq QP$ and

$$
\chi \triangleright L(\tilde{x}) \xrightarrow{\Omega} G \in QP \quad \land \quad y \in N \setminus \text{new}(\Omega)
$$

\[ \Downarrow \]

$$
\chi' \triangleright L(\tilde{x}\{y/x\}) \xrightarrow{\Omega\{y/x\}} G\{y/x\} \in QP
$$

where

$$
\chi' : \{\tilde{x}\} \setminus \{x\} \cup \{y\} \rightarrow S \\
\chi'(z) = \begin{cases} 
\chi(z), & z \in \{\tilde{x}\} \setminus \{x, y\} \\
\chi(x) + \chi(y), & z = y \land y \in \tilde{x} \\
\chi(x), & z = y \land y \not\in \{\tilde{x}\}
\end{cases}
$$

**SHReQ rewriting system:** $(QP, \Gamma \vdash G)$
Graph transitions

(REN)
\[ \chi \triangleright L(\tilde{x}) \xrightarrow{\Omega} G \in \mathcal{QP} \quad \rho = \text{mgu}(\Omega) \quad \bigwedge_{x \in \text{dom}(\chi)} \chi(x) \leq \Gamma(x) \]

\[ \Gamma \vdash L(\tilde{x}) \xrightarrow{\Omega} \Gamma_\Omega \vdash (\nu \; Z)(G\rho) \]

(COM)
\[ \Gamma_1 \vdash G_1 \xrightarrow{\Lambda_1} \Gamma'_1 \vdash G'_1 \quad \Gamma_2 \vdash G_2 \xrightarrow{\Lambda_2} \Gamma'_2 \vdash G'_2 \quad \rho = \text{mgu}(\Lambda_1 \cup \Lambda_2) \quad \bigwedge_{x \in \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2)} \Gamma_1(x) = \Gamma_2(x) \]

\[ \Gamma_1 \cup \Gamma_2 \vdash G_1 \mid G_2 \xrightarrow{\Lambda_1 \cup \Lambda_2} (\Gamma_1 \cup \Gamma_2)_{(\Lambda_1 \cup \Lambda_2)} \vdash (\nu \; Z)(G'_1 \mid G'_2)\rho \]

where \( Z = \text{new}(\Omega) \setminus \text{new}(\Omega) \)
Graph transitions

\[(\text{REN})\]
\[
\chi \triangleright L(\bar{x}) \xrightarrow{\Omega} G \in \mathcal{QP}
\]
\[
\rho = \text{mgu}(\Omega)
\]
\[
\bigwedge_{x \in \text{dom}(\chi)} \chi(x) \leq \Gamma(x)
\]
\[
\Gamma \vdash L(\bar{x}) \xrightarrow{\Omega} \Gamma_{\Omega} \vdash (\nu Z)(G\rho)
\]

\[(\text{COM})\]
\[
\Gamma_1 \vdash G_1 \xrightarrow{\Lambda_1} \Gamma_1' \vdash G_1'
\]
\[
\Gamma_2 \vdash G_2 \xrightarrow{\Lambda_2} \Gamma_2' \vdash G_2'
\]
\[
\bigwedge_{x \in \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2)} \Gamma_1(x) = \Gamma_2(x)
\]
\[
\Gamma_1 \cup \Gamma_2 \vdash G_1 \mid G_2 \xrightarrow{\Lambda_1 \cup \Lambda_2} (\Gamma_1 \cup \Gamma_2)_{(\Lambda_1 \cup \Lambda_2)} \vdash (\nu Z)(G_1' \mid G_2')\rho
\]

where \(Z = \text{new}(\Omega) \setminus \text{new}(\Omega)\)
Induced communication functions

Let $\rho = \text{mgu}(\Omega)$. The communication function induced by $\Omega$ is the function $\Omega : \text{dom}(\Omega) \to \mathcal{R}$ defined as

$$
\Omega(x) = \begin{cases} 
(t, \tilde{y} \rho), & t = \prod_{(x,s,\tilde{y}) \in \Omega \otimes x} s \notin \text{Sync} \\
(t, \langle \rangle), & t = \prod_{(x,s,\tilde{y}) \in \Omega \otimes x} s \in \text{Sync}
\end{cases}
$$

Basically, $\Omega(x)$ yields the synchronisation of requirements in $\Omega \otimes x$ according to the c-semiring product.

The weighting function induced by $\Gamma$ and $\Omega$ is

$$
\Gamma_\Omega : \text{dom}(\Gamma) \to S,
$$

$$
\Gamma_\Omega(x) = \begin{cases} 
1, & x \in \text{new}(\Omega) \\
\Gamma(x), & |\Omega \otimes x| = 1 \\
\Gamma_\Omega(x) = \Omega(x) \downarrow_1, & \text{otherwise}
\end{cases}
$$

The weighting function computes the new weights of graphs after the synchronisations induced by $\Omega$. 
Applying SHReQ’s semantics

Let’s compose broadcast and priority c-semirings:

\[
\begin{align*}
Sync_{BP} &= \{1\} = \{(1, \infty)\} \\
Fin_{BP} &= \{1\} \cup \{(\overline{a}, n) | \overline{a} \in W, n > 0\} \\
NoSync_{BP} &= \{0 = \langle 0, 0 \rangle, \perp \} \cup \{(a, 0) | a \in W\} \cup \{(0, n) | n \in \mathbb{N}\}
\end{align*}
\]

- The only value in \(Sync_{BP}\) is \(1\)
- \(Fin_{BP}\) are all coactions together with any valid priority \(n > 0\)
- \(NoSync_{BP}\) contains all pairs with at least one “zero” in their components.
Selecting productions

The initial graph

Checking alarm

Sending alarm

\[ x_1, x_2 : 0 \rightarrow R(x_1, x_2) \frac{(x_1, 1, \langle \rangle) (x_2, (\overline{a}, \infty), \langle \rangle)}{R^{ra}(x_1, x_2)} \]

\[ y_1 : 0 \rightarrow U_n(y_1) \frac{(y_1, (a, n), \langle \rangle)}{U_n^{wa}(y_1)} \]
Selecting productions

Checking alarm

Sending alarm

\[(\alpha, 1) \ast (\alpha, 3) \ast (\overline{\alpha}, \infty) = (\overline{\alpha}, 1)\]
A SHReQ synchronisation

\[
\begin{align*}
\text{r: 0} & \triangleright U_1(r) \xrightarrow{(r, (\alpha, 1), \langle \rangle)} U_1^{wa}(r) \{ r/y_1 \} \\
\text{r: 1} & \vdash U_1(r) \xrightarrow{(r, (\alpha, 1), \langle \rangle)} r: 1 \vdash U_1^{wa}(r) \\
\text{w, r: 0} & \triangleright R(w, r) \xrightarrow{(r, (\alpha, \infty), \langle \rangle)} R^{\alpha}(w, r) \{ w/x_1, r/x_2 \} \\
\text{w, r: 1} & \vdash R(w, r) \xrightarrow{(w, 1, \langle \rangle)} w, r, z: 1 \vdash R^{\alpha}(w, r) \\
\text{w, r: 1} & \vdash U_1(r) \mid R(w, r) \xrightarrow{(r, (\alpha, 1), \langle \rangle)} w, z: 1, r: (\overline{\alpha}, 1) \vdash U_1^{wa}(r) \mid R^{\alpha}(w, r) \\
\text{r: 0} & \triangleright U_3(r) \xrightarrow{(r, (\alpha, 3), \langle \rangle)} U_3^{wa}(r) \{ r/y_2 \} \\
\text{r: 1} & \vdash U_3(r) \xrightarrow{(r, (\alpha, 3), \langle \rangle)} r: 1 \vdash U_3^{wa}(r) \\
\text{w, r: 1} & \vdash U_1(r) \mid R(w, r) \mid U_3(r) \xrightarrow{(r, (\overline{\alpha}, 1), \langle \rangle)} w, z: 1, r: (\overline{\alpha}, 1) \vdash U_1^{wa}(r) \mid R^{\alpha}(w, r) \mid U_3^{wa}(r)
\end{align*}
\]
Exploiting the mgu

Sending ambulance assistance

\[ x \bullet 1 \langle \rangle \quad \bullet x \]

\[ \begin{array}{c}
S \\
\rightarrow \\
w \bullet (\alpha, \infty) \langle x \rangle \\
\rightarrow \\
x, w : 0 \triangleright S(x, w) \frac{(x, 1 \langle \rangle)}{(w, (\alpha, \infty), \langle x \rangle)} \rightarrow S(x, w)
\end{array} \]

Forwarding alarm

\[ w \bullet (\overline{\alpha}, \infty) \langle u \rangle \quad w \bullet \]

\[ \begin{array}{c}
R^{ra} \\
\rightarrow \\
r \bullet (\overline{\alpha}, \infty) \langle u \rangle \\
\rightarrow \\
w : 0, r : (\overline{\alpha}, 0) \triangleright R^{ra} (w, r) \frac{(w, (\overline{\alpha}, \infty), \langle u \rangle)}{(r, (\overline{\alpha}, \infty), \langle u \rangle)} \rightarrow R(w, r)
\end{array} \]

Receiving ambulance assistance

\[ r \bullet (\alpha, n) \langle z \rangle \quad r \bullet \]

\[ \begin{array}{c}
U^{wa}_n \\
\rightarrow \\
z \bullet \\
r : (\overline{\alpha}, n) \triangleright U^{wa}_n (r) \frac{(r, (\alpha, n), \langle z \rangle)}{U^{wa}_n (z)} \rightarrow U^{wa}_n (z)
\end{array} \]
Exploiting the mgu

\[ \rho = x/u, x/z \]

### Sending ambulance assistance

\[ x \cdot 1() \quad \rightarrow \quad x \]

\[ \begin{array}{c}
S \\
\rightarrow \\
S
\end{array} \]

\[ w \cdot (\alpha, \infty)\langle x \rangle \quad \rightarrow \quad w \]

\[ x, w : 0 \triangleright S(x, w) \otimes x, (\alpha, \infty), \langle x \rangle \quad \rightarrow \quad S(x, w) \]

### Forwarding alarm

\[ w \cdot (\bar{\alpha}, \infty)\langle u \rangle \quad \rightarrow \quad w \cdot \]

\[ \begin{array}{c}
R^{ra} \\
\rightarrow \\
R
\end{array} \]

\[ r \cdot (\bar{\alpha}, \infty)\langle u \rangle \quad \rightarrow \quad r \cdot \]

\[ w : 0, r : (\bar{\alpha}, 0) \triangleright R^{ra}(w, r) \otimes \langle w, (\bar{\alpha}, \infty), \langle u \rangle \rangle \quad \rightarrow \quad R(w, r) \]

### Receiving ambulance assistance

\[ r \cdot (\alpha, n)\langle z \rangle \quad \rightarrow \quad r \cdot \]

\[ \begin{array}{c}
U^{ua} \quad \rightarrow \\
U_{n}^{ua}
\end{array} \]

\[ z \cdot \]

\[ r : (\bar{\alpha}, n) \triangleright U^{wa}_{n}(r) \otimes \langle r, (\alpha, n), \langle z \rangle \rangle \quad \rightarrow \quad U_{n}^{ua}(z) \]
Conclusions

“Run, rabbit run
Dig that hole, forget the sun
And when at last the work is done
Don’t sit down it’s time to dig another one

(Breathe, Roger Waters)
Final Remarks

- We have presented SHReQ’s syntax and semantics
  - The main features are
    - general multiparty synchronisations c-semiring based
    - SHReQ uses application level QoS as coordination mechanism of distributed activities
  - Surprisingly, they fit the design principles in [Mil96]

We plan to

- Validate our design choice at the light of [Mil96]
- Develop formal methods based on SHReQ for specifying distributed applications that can bargain their non-functional requirements
- Hopefully, develop verification techniques (e.g., model checking) on SHReQ [FL04]

...
References


[HIM00] Dan Hirsch, Paola Inverardi, and Ugo Montanari. Reconfiguration of Software Architecture Styles


[MR96] Ugo Montanari and Francesca Rossi. Graph rewriting and constraint solving for modelling distributed systems with synchronization. In P. Ciancarini and

