Architectural Design Rewriting

R. Bruni\textsuperscript{1}, A. Lluch Lafuente\textsuperscript{1}, U. Montanari\textsuperscript{1}, E. Tuosto\textsuperscript{2}

\textsuperscript{1}Department of Computer Science, University of Pisa
\textsuperscript{2}Department of Computer Science, University of Leicester

14 June 2007
Outline

Introduction

Architectural Design Rewriting

Conclusion
Premises and Promises

Premises

D5.3a Reconfigurations preserving architectural types and shapes.

Promises

“\textit{In future, results of Deliverable 5.3a will be applied to help in deducing rules for reconfiguration.}”

- D6.1a

“We will develop a formal model for […] reconfiguration …”

- Wiki’s T5.3 description

“The objective of WP5 is to provide rigorous mathematical foundations for combining services, including […] reconfiguration.”

- Wiki’s WP5 objective
Premises and Promises

**Premises**

D5.3a Reconfigurations preserving architectural types and shapes.

**Promises**

“In future, results of Deliverable 5.3a will be applied to help in deducing rules for reconfiguration.”

- D6.1a

“We will develop a formal model for […] reconfiguration …”

- Wiki’s T5.3 description

“The objective of WP5 is to provide rigorous mathematical foundations for combining services, including […] reconfiguration.”

- Wiki’s WP5 objective
The recipe

### Ingredients
- 20mg (Dynamic) Software Architectures;
- 3 Architectural Styles;
- 10dl Reconfigurations;
- Some Graphs;
- QoS (at will).

### Preparation
Put the ingredients together and mix.
What are (software) architectures?

The experts say...

“... the structure of the components of a program / system, their interrelationship, and principles and guidelines governing their design and evolution over time.”

- D. Garlan & D. Perry, 1995
A suitable model for architectures

Graphs as model of typical architectural models
- E.g. Components and Connectors.

Graphs as model of Sensoria languages
- SRML diagrams (see SRML-P(v1.3)).
- Graphical encodings of process calculi.
A suitable model for architectures

Graphs as model of typical architectural models
- E.g. Components and Connectors.

![Diagram showing components, connectors, and ports]

Graphs as model of Sensoria languages
- SRML diagrams (see SRML-P(v1.3)).
- Graphical encodings of process calculi.
A style is . . .

“...a set of patterns or rules for creating one or more architectures in a consistent fashion.”

- IEEE standard 1471

Roughly... Style = Vocabulary + Rules

Benefits?
Understanding, Reuse, Construction, etc.
## Architectural Styles

A style is . . .

“. . . a set of patterns or rules for creating one or more architectures in a consistent fashion.”

- IEEE standard 1471

Roughly... Style = Vocabulary + Rules

### Benefits?

Understanding, Reuse, Construction, etc.
## Models for styles

### As a language
- Implicit style rules.
- Intuitive style-driven design.
- E.g. Graph grammars approaches [Le Métayer, 1996]

### As additional constraints
- Explicit style rules.
- Complex structural rules easier to express.
- E.g. Alloy-based approaches [Garlan 2006].
Models for styles

### As a language

- Implicit style rules.
- Intuitive style-driven design.
- E.g. Graph grammars approaches [Le Métayer, 1996]

### As additional constraints

- Explicit style rules.
- Complex structural rules easier to express.
- E.g. Alloy-based approaches [Garlan 2006].
## Reconfigurations

<table>
<thead>
<tr>
<th>What?</th>
<th>A reconfiguration is a change in a (dynamic) architecture.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why?</td>
<td>Security policies, load balancing, mobility, QoS assurance, etc.</td>
</tr>
<tr>
<td>E.g.?</td>
<td>Components joining/leaving the system, binding, wrapping, etc.</td>
</tr>
</tbody>
</table>
# Reconfigurations

## What?
A reconfiguration is a change in a (dynamic) architecture.

## Why?
Security policies, load balancing, mobility, QoS assurance, etc.

## E.g.?
Components joining/leaving the system, binding, wrapping, etc.
## Reconfigurations

<table>
<thead>
<tr>
<th>What?</th>
<th>A reconfiguration is a change in a (dynamic) architecture.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why?</td>
<td>Security policies, load balancing, mobility, QoS assurance, etc.</td>
</tr>
<tr>
<td>E.g.?</td>
<td>Components joining/leaving the system, binding, wrapping, etc.</td>
</tr>
</tbody>
</table>
Outline

Introduction

Architectural Design Rewriting

Conclusion
What is ADR?

ADR = Term Rewriting + Designs.
Designs = Typed Graphs with Interfaces.

Goal

A unifying model for ...

- Design;
- Execution;
- Reconfiguration.
What is ADR?

ADR = Term Rewriting + Designs.
Designs = Typed Graphs with Interfaces.

Goal

A unifying model for . . .

- Design;
- Execution;
- Reconfiguration.
A Client-Server Example. Sorts

Sorts (Vocabulary of Architectural Elements)

The sorts of the client server style are

- **Ports (nodes):** •.
- **Component types (edges, designs)**
  
  $C \rightarrow \bullet \quad S$

- **Actual components (edges)**
  
  client $\rightarrow \bullet$
  
  \(\bullet \leftrightarrow \text{connector} \rightarrow \bullet\)
  
  \(\bullet \leftarrow \text{orchestrator} \rightarrow \bullet\)
  
  \(\bullet \leftarrow \text{server} \rightarrow \bullet\)
A Client-Server Example. Sorts

Sorts (Vocabulary of Architectural Elements)

The sorts of the client server style are

- **Ports (nodes):** •.
- **Component types (edges, designs)**
  
  \[
  C \rightarrow • \quad S
  \]

- **Actual components (edges)**

  \[
  \text{client} \rightarrow • \quad • \leftarrow \text{connector} \rightarrow • \\
  • \leftarrow \text{orchestrator} \rightarrow • \quad • \leftarrow \text{server}
  \]
A Client-Server Example. Sorts

Sorts (Vocabulary of Architectural Elements)

The sorts of the client server style are

- Ports (nodes): ⋅.
- Component types (edges, designs)
  
    \[
    C \rightarrow ⋅ \quad \text{S}
    \]

- Actual components (edges)
  
    \[
    \text{client} \rightarrow ⋅ \quad ⋅ \leftarrow \text{connector} \rightarrow ⋅ \\
    ⋅ \leftarrow \text{orchestrator} \rightarrow ⋅ \quad ⋅ \leftarrow \text{server}
    \]
A design is a typed graph with an edge as interface.

... is a design of type $C$. 
A Client-Server Example. Operations

**system**: $C \rightarrow S$

- $S$ → $C$ → $\bullet$ → server

**osystem**: $C \rightarrow S$

- $S$ → $C$ → $\bullet$ ← orchestrator → $\bullet$ ← server
A Client-Server Example. Operations

\[
\text{system: } C \rightarrow S
\]

\[
\begin{array}{c}
S \\
C \rightarrow \bullet \leftarrow \text{server}
\end{array}
\]

\[
\text{osystem: } C \rightarrow S
\]

\[
\begin{array}{c}
S \\
C \rightarrow \bullet \leftarrow \text{orchestrator} \rightarrow \bullet \leftarrow \text{server}
\end{array}
\]
A Client-Server Example. Operations

\[ \text{cclient} : \rightarrow \text{C} \]

\[ \text{client} : \rightarrow \text{C} \]
A Client-Server Example. Operations

cclient → C

client → C

Architectural Design Rewriting
A Client-Server Example. Operations

clients: \( C \times C \rightarrow C \)

noclient: \( \rightarrow C \)
A Client-Server Example. Operations

**clients**: $C \times C \rightarrow C$

[Diagram of clients operation]

**noclient**: $\rightarrow C$

[Diagram of noclient operation]
A Client-Server Example. Values (again)

Values (evaluation of terms)

The value of clients(cclient, cclient) is ...
A Client-Server Example. Reconfiguration rules

**join(Y)**

\[ X \rightarrow \text{clients}(X, Y) \]

**leave(Y)**

\[ \text{clients}(X, Y) \rightarrow X \]

**orchestrate**

\[ \text{system}(X) \rightarrow \text{osystem}(X) \]
A Client-Server Example. Reconfiguration rules

**join(Y)**

\[ X \rightarrow \text{clients}(X, Y) \]

**leave(Y)**

\[ \text{clients}(X, Y) \rightarrow X \]

**orchestrate**

\[ \text{system}(X) \rightarrow \text{osystem}(X) \]
A Client-Server Example. Reconfiguration rules

**join(Y)**

\[ X \rightarrow \text{clients}(X, Y) \]

**leave(Y)**

\[ \text{clients}(X, Y) \rightarrow X \]

**orchestrate**

\[ \text{system}(X) \rightarrow \text{osystem}(X) \]
A Client-Server Example. A reconfiguration

Applying rule orchestrate...

If we have system(clients(cclient, cclient))...

...X is matched with clients(cclient, cclient)...

...and our system becomes osystem(clients(cclient, cclient)).
A Client-Server Example. A reconfiguration

Applying rule *orchestrate*...

If we have system(clients(cclient, cclient))... 

...X is matched with clients(cclient, cclient)... 

...and our system becomes osystem(clients(cclient, cclient)).
A Client-Server Example. A reconfiguration

Applying rule orchestrate...

If we have system(clients(cclient, cclient))...

\[ S \]

\[ ? \text{client} \rightarrow \bullet \leftarrow \text{connector} \rightarrow \bullet \leftarrow \text{server} \]

\[ ? \text{client} \rightarrow \bullet \leftarrow \text{connector} \]

\[ X \] is matched with clients(cclient, cclient)...

... and our system becomes osystem(clients(cclient, cclient)).

\[ S \]

\[ ? \text{client} \rightarrow \bullet \leftarrow \text{connector} \rightarrow \bullet \leftarrow \text{orchestrator} \rightarrow \bullet \leftarrow \text{server} \]

\[ ? \text{client} \rightarrow \bullet \leftarrow \text{connector} \]
A Client-Server Example. More reconfigurations

disconnect

Labeled rule to model individual disconnection:

\[
\text{client}^{\text{disc}} \rightarrow \text{client}
\]

Dummy labeled rule:

\[
\text{client}^{\text{disc}} \rightarrow \text{client}
\]

Conditional labeled rule for complex disconnections:

\[
\frac{X^{\text{disc}} \rightarrow X'}{\text{clients}(X, Y)^{\text{disc}} \rightarrow \text{clients}(X', Y')}
\]

\[
\frac{Y^{\text{disc}} \rightarrow Y'}{\text{clients}(X, Y)^{\text{disc}} \rightarrow \text{clients}(X', Y')}
\]
**A Client-Server Example. More reconfigurations**

**disconnect**

Labeled rule to model individual disconnection:

\[ c_{\text{client}} \xrightarrow{\text{disc}} \text{client} \]

Dummy labeled rule:

\[ \text{client} \xrightarrow{\text{disc}} \text{client} \]

Conditional labeled rule for complex disconnections:

\[
\frac{\begin{array}{c} X \xrightarrow{\text{disc}} X' \\
Y \xrightarrow{\text{disc}} Y' \end{array}}{	ext{clients}(X, Y) \xrightarrow{\text{disc}} \text{clients}(X', Y')}\]
A Client-Server Example. More reconfigurations

disconnect

Labeled rule to model individual disconnection:

\[ \text{cclient} \xrightarrow{\text{disc}} \text{client} \]

Dummy labeled rule:

\[ \text{client} \xrightarrow{\text{disc}} \text{client} \]

Conditional labeled rule for complex disconnections:

\[
\frac{\mathcal{X} \xrightarrow{\text{disc}} \mathcal{X}' \quad \mathcal{Y} \xrightarrow{\text{disc}} \mathcal{Y}'}{
\text{clients}((\mathcal{X}, \mathcal{Y}) \xrightarrow{\text{disc}} \text{clients}((\mathcal{X}', \mathcal{Y}'))}
\]
A Client-Server Example. Another reconfiguration

Applying rule disconnect...

If we have osystem(clients(cclient, cclient))...

...we obtain osystem(clients(client, client)).
A Client-Server Example. Another reconfiguration

Applying rule disconnect...

If we have \( \text{osystem}(\text{clients}(c_{\text{client}}, c_{\text{client}})) \ldots \)

\[ S \]

\[ \quad \text{client} \rightarrow \bullet \leftarrow \text{connector} \rightarrow \bullet \leftarrow \text{orchestrator} \rightarrow \bullet \leftarrow \text{server} \]

\[ \quad \text{client} \rightarrow \bullet \leftarrow \text{connector} \]

\ldots we obtain \( \text{osystem}(\text{clients}(\text{client, client})) \).

\[ S \]

\[ \quad \text{client} \rightarrow \bullet \leftarrow \text{orchestrator} \rightarrow \bullet \leftarrow \text{server} \]

\[ \quad \text{client} \rightarrow \bullet \leftarrow \text{orchestrator} \rightarrow \bullet \leftarrow \text{server} \]

\[ \quad \text{client} \]
# Constrained Designs

## Designs with Constraints

- Nodes as QoS attributes.
- Edges as (c-semiring based) constraints.

## Purpose

- Postpone design decisions.
- Trigger reconfigurations.
- Measure reconfiguration alternatives.
Constrained Designs

**Designs with Constraints**

- Nodes as QoS attributes.
- Edges as (c-semiring based) constraints.

**Purpose**

- Postpone design decisions.
- Trigger reconfigurations.
- Measure reconfiguration alternatives.
ADR in detail

ADR is . . .

- Sorts: Vocabulary, Types (Edge and node labels).
- Values: Designs (graphs with interfaces).
- Operations: grammar-like style rules.
- Terms: proofs of construction.
- Terms (with variables): partial Designs.
- Axioms: properties of operations.
- Membership equations: additional style rules.
- Rewriting rules: behaviour, reconfigurations.
- Rewriting strategies: style conformance, style analysis, etc.
What next?

Ongoing...

- Implementation in Maude.
- Style for a graphical encoding of the $\pi$-calculus.
- Styles for SHR variants.

Future... (Sensorize the Approach)

- Not just symbolic types: service, behavioural, spatial, etc.
- Application to SRML architectures.
- Case Study / Scenario.
What next?

Ongoing...

- Implementation in Maude.
- Style for a graphical encoding of the $\pi$-calculus.
- Styles for SHR variants.

Future... (Sensorize the Approach)

- Not just symbolic types: service, behavioural, spatial, etc.
- Application to SRML architectures.
- Case Study / Scenario.
What next?

Ongoing...

- Implementation in Maude.
- Style for a graphical encoding of the $\pi$-calculus.
- Styles for SHR variants.

Future... (Sensorize the Approach)

- Not just symbolic types: service, behavioural, spatial, etc.
- Application to SRML architectures.
- Case Study / Scenario.
## What next?

### Ongoing...
- Implementation in Maude.
- Style for a graphical encoding of the \( \pi \)-calculus.
- Styles for SHR variants.

### Future... (Sensorize the Approach)
- Not just symbolic types: service, behavioural, spatial, etc.
- Application to SRML architectures.
- Case Study / Scenario.
What next?

**Ongoing...**
- Implementation in Maude.
- Style for a graphical encoding of the $\pi$-calculus.
- Styles for SHR variants.

**Future... (Sensorize the Approach)**
- Not just symbolic types: service, behavioural, spatial, etc.
- Application to SRML architectures.
- Case Study / Scenario.
What next?

Ongoing...
- Implementation in Maude.
- Style for a graphical encoding of the $\pi$-calculus.
- Styles for SHR variants.

Future... (Sensorize the Approach)
- Not just symbolic types: service, behavioural, spatial, etc.
- Application to SRML architectures.
- Case Study / Scenario.
What next?

**Ongoing...**
- Implementation in Maude.
- Style for a graphical encoding of the $\pi$-calculus.
- Styles for SHR variants.

**Future... (Sensorize the Approach)**
- Not just symbolic types: service, behavioural, spatial, etc.
- Application to SRML architectures.
- Case Study / Scenario.
Further reading

Two drafts

- Style-Based Reconfigurations of Software Architectures with QoS Constraints.
I got the idea but I have some...

▶ Questions?
▶ Remarks?
▶ Criticism?
▶ Suggestions?