Hierarchical Design Rewriting with Maude

Alberto Lluch Lafuente, Roberto Bruni, Ugo Montanari

Department of Computer Science
University of Pisa
{bruni,lafuente,ugo}@di.unipi.it

Software Engineering for
Service-Oriented Overlay Computers

7th Int’l Workshop on Rewriting Logic and its Applications
Budapest, March 29-30, 2008
Sensoria (Poster Collage)

Software Engineering for Service-Oriented Overlay Computers  www.sensoria-ist.eu

develops

semantically well-founded languages, novel theories, methods and tools for constructing and analysing the new generation of high-quality service-oriented systems

integrates

foundational theories, techniques, and methods with pragmatic software engineering

researches

- linguistic primitives for modelling and programming service-oriented systems
- qualitative and quantitative analysis methods for global services
- development and deployment techniques for systems services

offers

- model-driven approach for service-oriented software engineering
- modelling of service-oriented systems
- analysis of behaviour, security and quality of service properties
- suite of tools and techniques for deploying service-oriented systems
- reengineering legacy software into services

case studies

in automotive, finance, telecommunications and and e-learning domains

List of partners

Coordinator: Prof. Dr. Martin Wirsing, Ludwig-Maximilians-Universität München, Germany
Universität di Trento | University of Leicester | Warsaw University | TU Denmark at Lyngby | Università di Pisa
Università di Firenze | Università di Bologna | ISTI Pisa | Universidade de Lisboa | University of Edinburgh | ATX | Telecom Italia Lab | Imperial College London | FAST 6mbH | Budapest University of Technology and Economics S&N AG | University College London | Politecnico di Milano
Running Example

We want to design and analyse reconfigurable filter architectures:

- We allow to compose filters in sequence or parallel
- .. and forbid disconnected and cyclic parts.
- Some filters are (services) not known at design-time.
- Run-time reconfigurations are needed (e.g. to ensure QoS)
Some problems we face

How can we design such software architectures?

- Some solutions:
  - Drop & bind components, check, correct: tedious.
  - Bounded SAT (à la Alloy): no guidance, trial&error.

Disclaimer: some flaws of some solutions that still remain valid.
Some problems we face

How can we design such software architectures?
▶ Some solutions:
  ▶ Drop & bind components, check, correct: tedious.
  ▶ Bounded SAT (à la Alloy): no guidance, trial&error.

How can we define property-preserving reconfigurations?
▶ Some solutions:
  ▶ Show a theorem: manual.
  ▶ Model checking: undecidable in general.
  ▶ Monitor & Repair: no design-time guarantee.

Disclaimer: some flaws of some solutions that still remain valid.
Principles of ADR

Architectural Design Rewriting:

- Algebra of *design terms*
  - Type $T_\phi$ set of architectures that satisfy $\phi$.
  - Set of design productions (operations, inductive definitions).

- Domain of *Designs*
  - Designs: hierarchical graphs with interfaces (HDR).
  - Partial designs: designs with holes.

- Reconfiguration as *Rewriting*
  - Rewrite design terms (not designs) $d : T \rightarrow d' : T$.
  - Based on conditional term rewriting, SOS.
Principles of ADR

Architectural Design Rewriting:

- **Algebra of design terms**
  - Type $T_\phi$ set of architectures that satisfy $\phi$.
  - Set of design productions (operations, inductive definitions).

- **Domain of Designs**
  - Designs: hierarchical graphs with interfaces (HDR).
  - Partial designs: designs with holes.

- **Reconfiguration as Rewriting**
  - Rewrite design terms (not designs) $d : T \rightarrow d' : T$.
  - Based on conditional term rewriting, SOS.

No panacea: not everything can be modelled with ADR, but you should be happy if you manage to capture part of your problem.
Pipes-and-Filters (Designs)

Architectures as graphs:

- components are hyperedges (boxes),
- ports are tentacles (arrows),
- and connectors are nodes (circles),
- interfaces are types (blue boxes).

Implemented in modules

- GRAPH-*
- DESIGN-*
We define our style of pipes-and-filters in an inductive manner.

A filter is...
- A single filter
- 2 sequential filters
- 2 parallel filters

```plaintext
fmod FILTER-STYLE is
  sort Filter .
  op filter : -> Filter [...] .
  op seq : Filter Filter -> Filter [assoc...] .
  op par : Filter Filter -> Filter [...] .
endfm
```
Pipes-and-Filters (Interpreted Design Productions)

Interpretation of design productions:

▶ for each sort we have an interface type,
▶ e.g. for sort Filter, we have a Filter-labelled edge exposing two nodes,
▶ an operation is like a design, where some edges are arguments,
▶ and substitution means hyperedge replacement.
Pipes-and-Filters (Interpreted Design Productions)

```
filter : -> Filter

bypass : -> Filter

par : Filter Filter -> Filter

mux

Filter

Filter

dmux

seq : Filter Filter -> Filter

Filter

Filter

Powered by yFiles
Pipes-and-Filters (Interpreted Term)

$$\text{seq}(\text{filter}, \text{par}(\text{filter}, \text{filter}))$$

(before substitution)
Pipes-and-Filters (Interpreted Term)

\[
\text{seq}(\text{filter1}, \text{par}(\text{filter2}, \text{filter3}))
\]

(after substitution)
Pipes-and-Filters (Reconfiguration)

We define reconfigurations as rewrite rules:

\[
\begin{align*}
\text{filter} & \xrightarrow{2\text{seq}} \text{filter} \\
\text{x}_1 & \xrightarrow{2\text{seq}} \text{x}_3 \quad \text{x}_2 & \xrightarrow{2\text{seq}} \text{x}_4 \\
\text{seq}(\text{x}_1,\text{x}_2) & \xrightarrow{2\text{seq}} \text{seq}(\text{x}_3,\text{x}_4) \\
\text{x}_1 & \xrightarrow{2\text{seq}} \text{x}_3 \quad \text{x}_2 & \xrightarrow{2\text{seq}} \text{x}_4 \\
\text{par}(\text{x}_1,\text{x}_2) & \xrightarrow{2\text{seq}} \text{seq}(\text{x}_3,\text{x}_4)
\end{align*}
\]

mod FILTER-RECONFIGURATION is

\[
\begin{align*}
\text{rl} & : \text{filter} \Rightarrow \{2\text{seq}\}\text{filter} . \\
\text{cr1} & : \text{seq}(\text{x}_1,\text{x}_2) \Rightarrow \{2\text{seq}\}\text{seq}(\text{x}_3,\text{x}_4) \\
& \quad \text{if x}_1 \Rightarrow \{2\text{seq}\} \text{x}_3 /\ \text{x}_2 \Rightarrow \{2\text{seq}\} \text{x}_4 . \\
\text{cr1} & : \text{par}(\text{x}_1,\text{x}_2) \Rightarrow \{2\text{seq}\}\text{seq}(\text{x}_3,\text{x}_4) \\
& \quad \text{if x}_1 \Rightarrow \{2\text{seq}\} \text{x}_3 /\ \text{x}_2 \Rightarrow \{2\text{seq}\} \text{x}_4 .
\end{align*}
\]

endm
Pipes-and-Filters (Interpreted Reconfiguration)
Pipes-and-Filters (Modelling Activities)

A right-to-left reading of operations:
- results in a grammar to generate all possible architectures,
- simulates design-by-refinement,
- can be used for model finding.

```
mod FILTER-REFINEMENT is
  op Filter-nt : -> Filter [ctor] .
  rl : Filter-nt => bypass .
  rl : Filter-nt => filter .
  rl : Filter-nt => seq(Filt\text{-}er-nt,Filter-nt) .
  rl : Filter-nt => par(Filt\text{-}er-nt,Filter-nt) .
endm
```
Pipes-and-Filters (Property Specification)

Structural properties given...
▶ over design terms (e.g. à la VLRL),
▶ over designs (e.g. à la MSO).

Temporal properties
▶ over the state space of reconfigurations,
▶ as LTL formulae, strategies, etc..
Pipes-and-Filters (Quick Analysis Example)

We require some ordering constraints $\phi$ among filters.

Maude> srew FClient-nt using modelCheck($\phi$)
Solution 7
result FClient: wrap(par(filter(1), Mux-nt, Dmux-nt ...
Pipes-and-Filters (Quick Analysis Example)

We require some ordering constraints \( \phi \) among filters.

Maude> srew FClient-nt using modelCheck(\( \phi \))
Solution 7
result FClient: wrap(par(filter(1), Mux-nt, Dmux-nt ...)

Does the 7th solution preserve some other constraints \( \psi \)?

Maude> red modelCheck(sol7,[\( \psi \)]) .
result ModelCheckResult: counterexample...
Pipes-and-Filters (Quick Analysis Example)

We require some ordering constraints \( \phi \) among filters.

\[
\text{Maude> srew FClient-nt using modelCheck(\phi)} \\
\text{Solution 7} \\
\text{result FClient: wrap(par(filter(1), Mux-nt, Dmux-nt \ldots)}
\]

Does the 7th solution preserve some other constraints \( \psi \)?

\[
\text{Maude> red modelCheck(sol7,[]}\psi\text{)} . \\
\text{result ModelCheckResult:} \\
\text{counterexample...}
\]

We ask for an architecture satisfying \( \phi \) and preserving \( \psi \).

\[
\text{Maude> srew FClient-nt using modelCheck(} \phi \text{ \wedge } []\psi\text{)} \\
\text{Solution 3} \\
\text{result FClient: wrap(seq(filter(0), par(filter(1), \ldots)}
\]
Summary

What is ADR?

- A formal method for reconfigurable architectures.
- Based on term rewriting.
- Based on graphs (HDR).
- Supported by Maude.

What can I do ADR?

- Design software architectures respecting structural properties.
- Define property preserving, inductive reconfigurations.
- Analyse architectures (e.g. Model Finding, Model Checking).
Some Examples

Leg-o-motive Case Study

Network Topologies

Architectural Styles

Process Algebras

Service Modelling Languages
Some Pointers

- Links
  - http://sensoria.fast.de/
- Papers:
  - Hierarchical Design Rewriting [WRLA'08]
  - Service Oriented Architectural Design [TGC'07]
  - Style-Based Architectural Reconfigurations [EATCS]
- Mail
  - \{bruni,lafuente,ugo\}@di.unipi.it
ADR is a three-letter acronym that may refer to:

- Académie de Roberval, a school in Montreal, Canada
- Short for Accord européen relatif au transport international des marchandises dangereuses par route, also known as the European Agreement concerning the International Carriage of Dangerous Goods by Road
- Adiabatic Damagnetisation Refrigeration
- Adria Airways, an airline of Slovenia (ICAO code: ADR)
- Advanced Digital Radio Testing Service
- Advanced Dungeons & Rabbits, a Role Playing Game for phpBB
- Adverse drug reaction
- Airdrie railway station, United Kingdom (National Rail code: ADR)
- Alter Der Ruine, a power noise group from Tucson, Arizona
- Alternative Democratic Reform Party, a political party in Luxembourg
- Alternative dispute resolution
- American Depositary Receipt, a method of trading foreign stock
- Andrew Capital Air, a subsidiary of South African Airways (IATA code: ADR)
- Applied Data Research
- Artificial Disc Replacement
- Astra Digital Radio
- Australian Design Rules, a set of construction standards for road registered vehicles in Australia
- Automated Dialogue Replacement or Additional Dialogue Recording, also known as "dubbing"
- Average daily rate, a common lodging industry statistic
- Azerbaijan Democratic Republic

adr may also mean:

- The adr microformat, a sub-set of the hCard microformat.