Hierarchical Design Rewriting with Maude

Alberto Lluch Lafuente, Roberto Bruni, Ugo Montanari

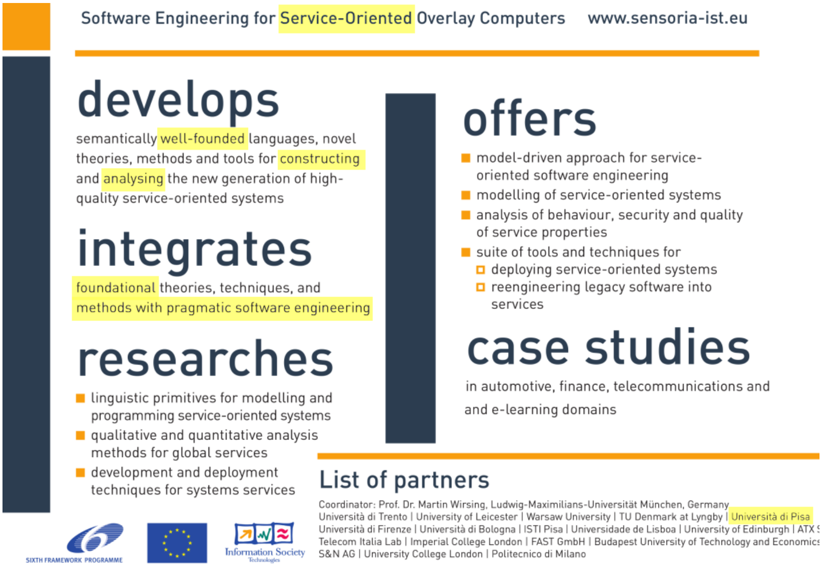
Department of Computer Science
University of Pisa

Software Engineering for
Service-Oriented Overlay Computers

{bruni,lafuente,ugo}@di.unipi.it

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
Università 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 S
Telecom Italia Lab | Imperial College London | FAST GmbH | Budapest University of Technology and Economics
S&N AG | University College London | Politecnico di Milano

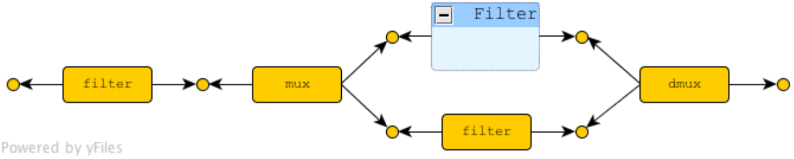
SIXTH FRAMEWORK PROGRAMME

Information Society Technologies

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.

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 **D**esign **R**ewriting:

- **A**lgebra of *design terms*
  - Type T_ϕ set of architectures that satisfy ϕ.
  - Set of design productions (operations, inductive definitions).
- **D**omain of **D***esigns*
  - Designs: hierarchical graphs with interfaces (HDR).
  - Partial designs: designs with holes.
- **R**econfiguration as **R**ewriting
  - Rewrite design terms (not designs) $d : T \rightarrow d' : T$.
  - Based on conditional term rewriting, SOS.

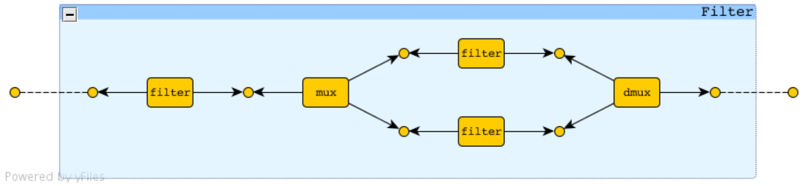
Principles of ADR

Architectural **D**esign **R**ewriting:

- **A**lgebra of *design terms*
  - Type T_ϕ set of architectures that satisfy ϕ.
  - Set of design productions (operations, inductive definitions).
- **D**omain of **D***esigns*
  - Designs: hierarchical graphs with interfaces (HDR).
  - Partial designs: designs with holes.
- **R**econfiguration as **R**ewriting
  - Rewrite design terms (not designs) $d : T \to d' : T$.
  - 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-*`

Pipes-and-Filters (Design Productions)

We define our style of pipes-and-filters in an inductive manner

A filter is...

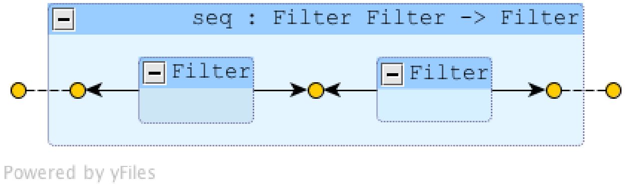
- A single filter
- 2 sequential filters
- 2 parallel filters

```
 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)

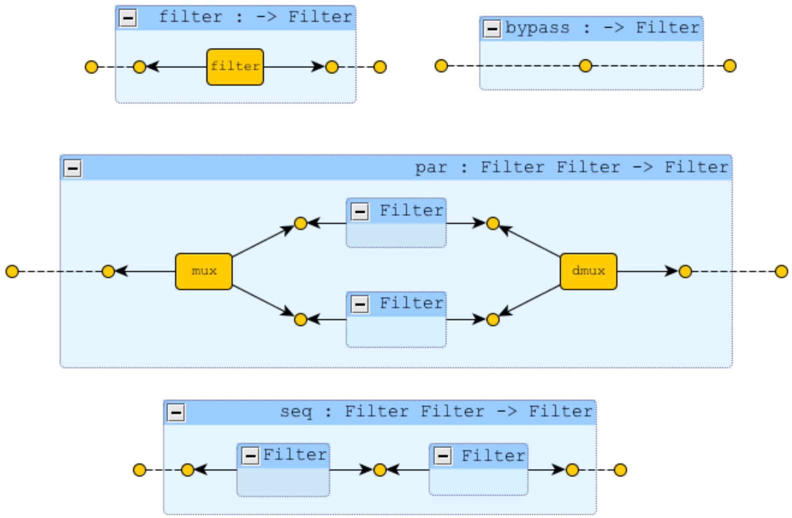
Interpretation of design productions:

- 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*.



```
fmod FILTER-DESIGN
```

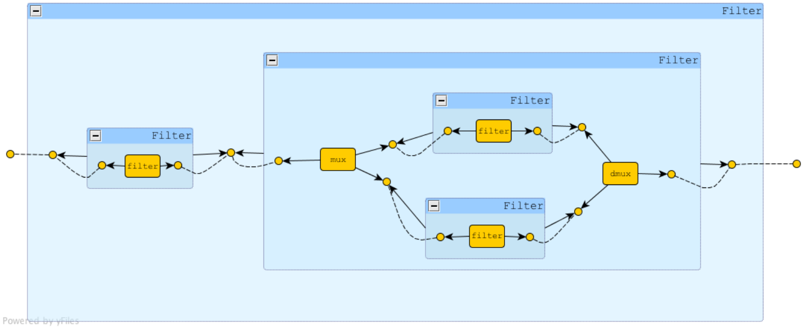
Pipes-and-Filters (Interpreted Design Productions)



Powered by yFiles

Pipes-and-Filters (Interpreted Term)

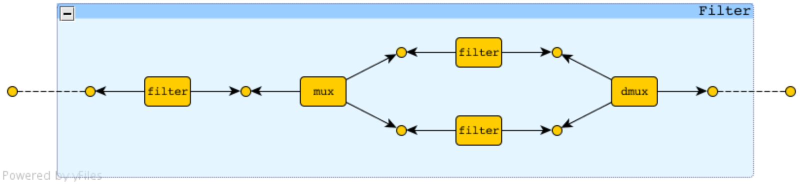
```
seq(filter,par(filter,filter))
```



(before substitution)

Pipes-and-Filters (Interpreted Term)

`seq(filter1,par(filter2,filter3))`



(after substitution)

Pipes-and-Filters (Reconfiguration)

We define reconfigurations as rewrite rules:

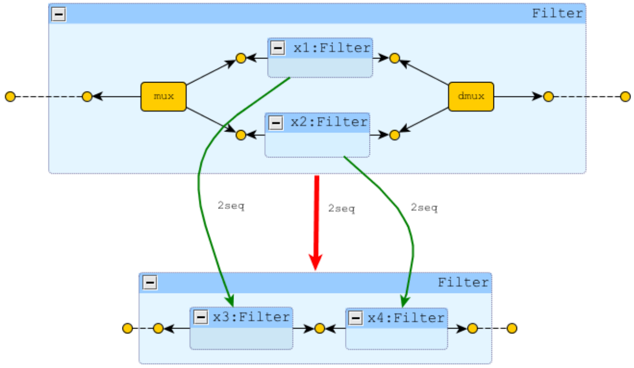
$$\texttt{filter} \xrightarrow{\texttt{2seq}} \texttt{filter} \qquad \frac{\texttt{x1} \xrightarrow{\texttt{2seq}} \texttt{x3} \quad \texttt{x2} \xrightarrow{\texttt{2seq}} \texttt{x4}}{\texttt{seq(x1,x2)} \xrightarrow{\texttt{2seq}} \texttt{seq(x3,x4)}}$$

$$\frac{\texttt{x1} \xrightarrow{\texttt{2seq}} \texttt{x3} \quad \texttt{x2} \xrightarrow{\texttt{2seq}} \texttt{x4}}{\texttt{par(x1,x2)} \xrightarrow{\texttt{2seq}} \texttt{seq(x3,x4)}}$$

Standard SOS-in-RL encoding

```
mod FILTER-RECONFIGURATION is
  rl : filter => {'2seq}filter .
  crl : seq(x1,x2) => {'2seq}seq(x3,x4)
   if x1 => {'2seq} x3 /\ x2 => {'2seq} x4 .
  crl : par(x1,x2) => {'2seq}seq(x3,x4)}
   if x1 => {'2seq} x3 /\ x2 => {'2seq} x4 .
endm
```

Pipes-and-Filters (Interpreted Reconfiguration)



Powered by yFiles

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(Filter-nt,Filter-nt) .
 rl : Filter-nt => par(Filter-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..

```
mod FILTER-PROP
mod MSO
```

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.

```
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...
```

We ask for an architecture satisfying phi and preserving psi.

```
Maude> srew FClient-nt using modelCheck(phi /\ []psi)
Solution 3
result FClient: wrap(seq(filter(0), par(filter(1), ...
```

Summary

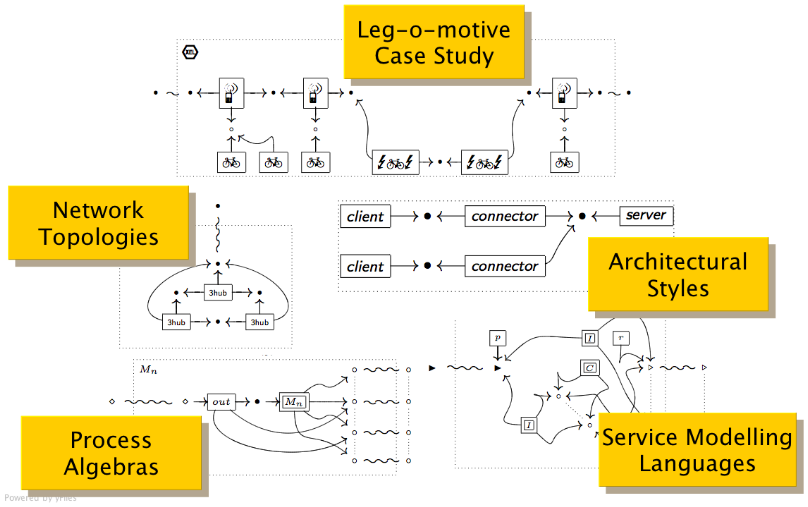
What is ADR?

- 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



Some Pointers

- Links
  - `http://www.albertolluch.com/adr.html`
  - `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`

Questions?

ADR is a three-letter acronym that may refer to:

- Académie de Roberval, a school in Montreal, Canada
- short for ***A****ccord européen relatif au transport international des marchandises* ***d****angereuses par* ***r****oute*, also known as the European Agreement concerning the International Carriage of Dangerous Goods by Road
- Adiabatic Demagnetisation 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
- Andrews Municipal Airport, located in South Carolina (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.