A service-oriented UML profile with formal support

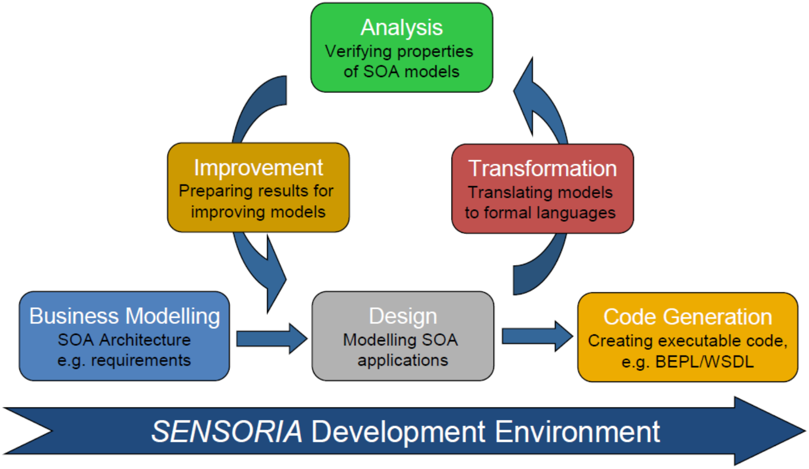
Roberto Bruni[1] Matthias Hölzl[3] Nora Koch[2,3]
Alberto Lluch Lafuente[1] Philip Mayer[3] Ugo Montanari[1]
Andreas Schroeder[3] Martin Wirsing[2]

[1]Dipartimento di Informatica, Università di Pisa
[2]Cirquent GmbH [3]Ludwig-Maximilians-Universität München
Software Engineering for Service-Oriented Overlay Computers (SENSORIA)

7th Int'l Joint Conference on Service Oriented Computing
Stockholm, November 23-27, 2009

INTRODUCTION

SENSORIA's Development Process


Analysis
Verifying properties of SOA models
Improvement
Preparing results for improving models
Transformation
Translating models to formal languages
Business Modelling
SOA Architecture e.g. requirements
Design
Modelling SOA applications
Code Generation
Creating executable code, e.g. BEPL/WSDL
SENSORIA Development Environment

UML4SOA

UML4SOA [KMH+07] offers a visual modelling language for Service-Oriented Applications:

- high-level front-end based on de-facto standards (UML2);
- minimalist extension of UML2 (as profiles);
- (model driven) transformations into formal languages.
- (model driven) transformations implementation languages.

UML4SOA Profiles

Profiles for domain specific aspects:

- behaviour;
- non-functional properties;
- reconfiguration;
- policies;
- requirements;

. . . and style-driven reconfigurations (this talk).

UML4SOA profile for style-driven reconfiguration

UML notation for a formal approach based on

- graphs as a model of architectural configuration;
- term rewriting as a model of reconfiguration.

UML4SOA profile for style-driven reconfiguration

UML notation for a formal approach based on

- graphs as a model of architectural configuration;
- term rewriting as a model of reconfiguration.

Why graphs?

- long tradition as a mathematical object for diagrams.

UML4SOA profile for style-driven reconfiguration

UML notation for a formal approach based on

- graphs as a model of architectural configuration;
- term rewriting as a model of reconfiguration.

Why graphs?

- long tradition as a mathematical object for diagrams.

Why term rewriting?

- long tradition as a model for system dynamics.

Reconfiguration Features of Services

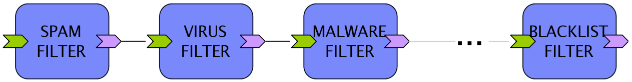
Usually, service descriptions regard functional or QoS aspects.

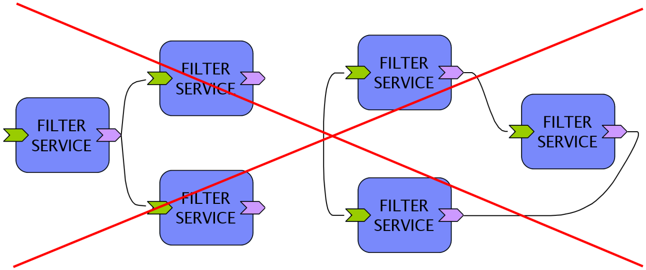
We focus on architectural reconfiguration features:

- to require services to be able to react to certain events with well-studied reconfigurations;
- to require services to have a certain well-studied shape which will drive the reconfiguration.

A simple example of style: filter chains

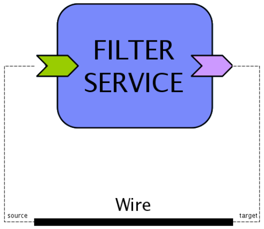
"filter services that can be combined as a linear chain"





Filter chains: UML-like approach

"A **Chain** *is an instance of the below diagram ..."*

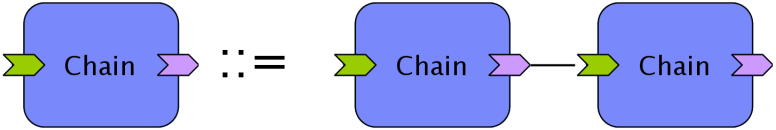


"... and further (OCL/SOL/...) constraints: connected, no cycle, no branching, ..."

$$\textbf{connected} \equiv \forall a, b.\forall X.((\forall x, y(y \in X \wedge z \in R(y,z) \rightarrow z \in X \\ \wedge \forall y.R(a,y) \rightarrow y \in X)) \rightarrow b \in X)$$

Filter chains: Generative approach

"A ***Chain*** *can be refined as two concatenated* ***Chains****"*

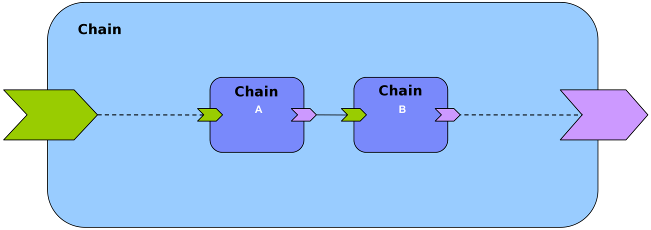


Architectural style as context-free (graph) grammar (e.g. [Le 98])

- Non-terminals play the role of styles (e.g. **Chain**);
- Grammar productions define the language of conformant architectures (e.g. **Chain** ::= **Chain** ; **Chain**).

Filter chains: Another generative approach

"The concatenation of two **Chain**s *forms a* **Chain***"*

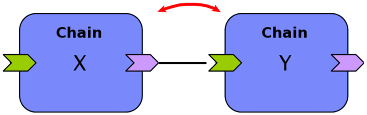


Architectural style as (graph) algebra (e.g. [BLMT08])

- Sorts play the role of styles (e.g. **Chain**);
- Operations represent the way of composing conformant architectures (e.g. $A; B$: **Chain** $\times$ **Chain** $\rightarrow$ **Chain**).

Architectural reconfiguration as rewrite rules

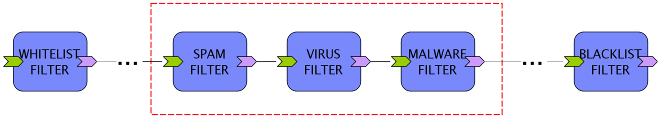
A simple rule for "swapping" chains: $x; y \rightarrow y; x$



This rule

Architectural reconfiguration as rewrite rules

A simple rule for "swapping" chains: $x; y \rightarrow y; x$

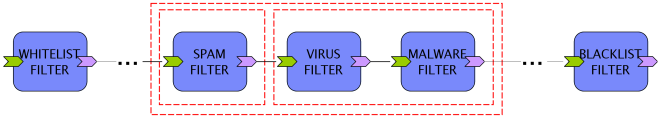


This rule

1. matches any (sub)chain s' of a chain s ;

Architectural reconfiguration as rewrite rules

A simple rule for "swapping" chains: $x;y \longrightarrow y;x$

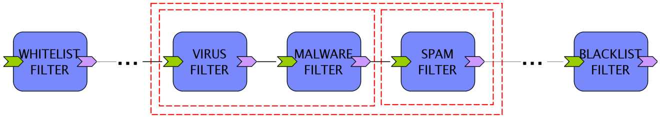


This rule

1. matches any (sub)chain s' of a chain s ;
2. divides s' in any two (sub)chains $x;y$;

Architectural reconfiguration as rewrite rules

A simple rule for "swapping" chains: $x; y \rightarrow y; x$

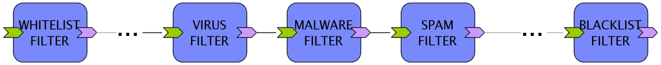


This rule

1. matches any (sub)chain s' of a chain s ;
2. divides s' in any two (sub)chains $x; y$;
3. builds s'' as $y; x$;

Architectural reconfiguration as rewrite rules

A simple rule for "swapping" chains: $x; y \rightarrow y; x$



This rule

1. matches any (sub)chain s' of a chain s ;
2. divides s' in any two (sub)chains $x; y$;
3. builds s'' as $y; x$;
4. replaces s' by s'' in s.

Some advantages of the operational approach

Design of style-conformant architectures

- Style-driven design-by-refinement: replace a variable (unspecified sub-component) by a term of the same type.
- alternative to
    - drop&bind components, check&correct: tedious, error prone;
    - model finding (à la Alloy): trial & error, no guidance.

Some advantages of the operational approach

Design of style-conformant architectures

- Style-driven design-by-refinement: replace a variable (unspecified sub-component) by a term of the same type.
- alternative to
  - drop&bind components, check&correct: tedious, error prone;
  - model finding (à la Alloy): trial & error, no guidance.

Style-preserving reconfigurations

- Style preservation immediate with rule $l : T \rightarrow r : T$.
- alternative to
  - prove theorems: ad-hoc, manual, limited re-use;
  - model checking: inefficient, undecidable in general;
  - monitor & repair: no guarantees at design-time;

Some advantages of the operational approach

Design of style-conformant architectures

- Style-driven design-by-refinement: replace a variable (unspecified sub-component) by a term of the same type.
- alternative to
  - drop&bind components, check&correct: tedious, error prone;
  - model finding (à la Alloy): trial & error, no guidance.

Style-preserving reconfigurations

- Style preservation immediate with rule $l : T \rightarrow r : T$.
- alternative to
  - prove theorems: ad-hoc, manual, limited re-use;
  - model checking: inefficient, undecidable in general;
  - monitor & repair: no guarantees at design-time;

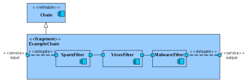
Rewrite engines support analysis

- membership to determine style conformance;
- exploration algorithms to find or check reconfiguration plans.

There are of course other pros and cons (see [BBGL08]).

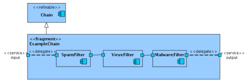
UML4SOA PROFILE

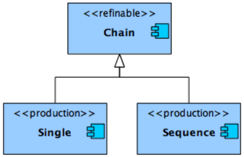
UML4SOA's profile main ingredients



- Fragment: a kind of internal structure diagram that describes an architectural configuration;

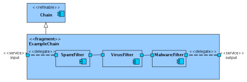
UML4SOA's profile main ingredients

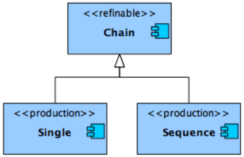


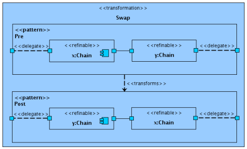


- Fragment: a kind of internal structure diagram that describes an architectural configuration;
- Patterns: a kind of class diagrams that define an architectural style in an inductive manner;

UML4SOA's profile main ingredients

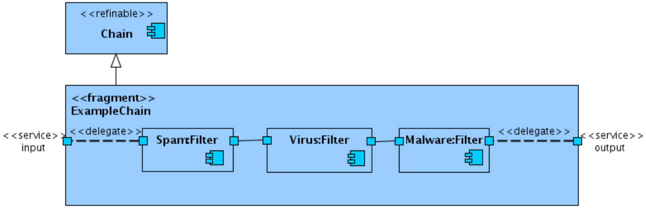






- Fragment: a kind of internal structure diagram that describes an architectural configuration;
- Patterns: a kind of class diagrams that define an architectural style in an inductive manner;
- Reconfiguration package: diagrams that specify reconfiguration rules.

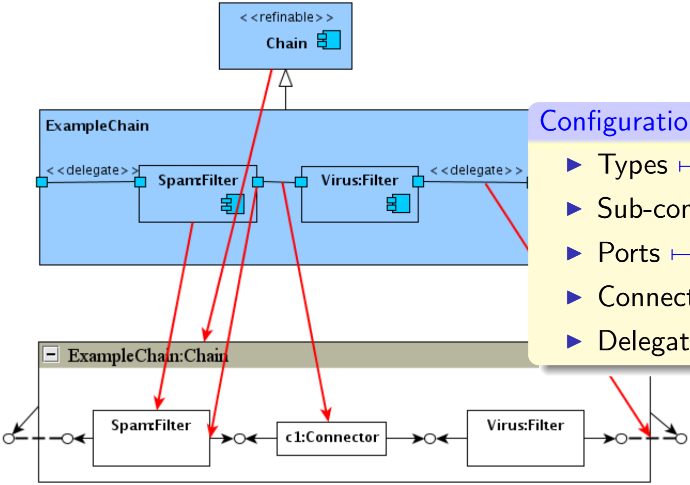
Configurations: Diagrams



Extended ≪fragment≫ internal structure diagrams:

- Define the internal structure of a (sub)system using
  - components (services);
  - ≪service≫ ports (required/provided service descriptions);
  - connectors (service references);
- ≪delegate≫ dependencies denote which internal ports play the role of external ports.

Configurations: Underlying Model



Configurations as Designs

- Types $\mapsto$ Types
- Sub-comps $\mapsto$ Edges
- Ports $\mapsto$ Tentacles
- Connectors $\mapsto$ Edges
- Delegates $\mapsto$ Interface

Configurations: Analysis

Does my architecture satisfy some given property?

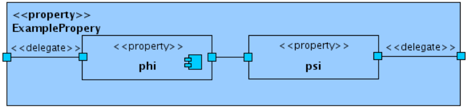
- Structural property expressed with some logic-based mechanism (OCL,MSO);

Configurations: Analysis

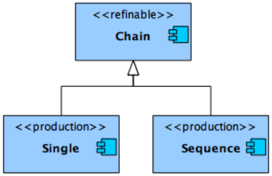
Does my architecture satisfy some given property?

- Structural property expressed with some logic-based mechanism (OCL,MSO);
- . . . or an ad-hoc spatial logic: the dual of the algebra.

Example: *"My* **Chain** *is made of two concatenated chains satisfying ϕ and ψ, respectively."* is expressed by $\phi\ ;\psi$.



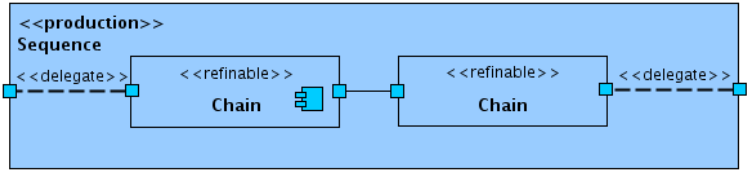
Architectural Styles: Diagrams



Patterns determine the style-conformant compositions:

- ≪refineable≫ component: an architectural type.

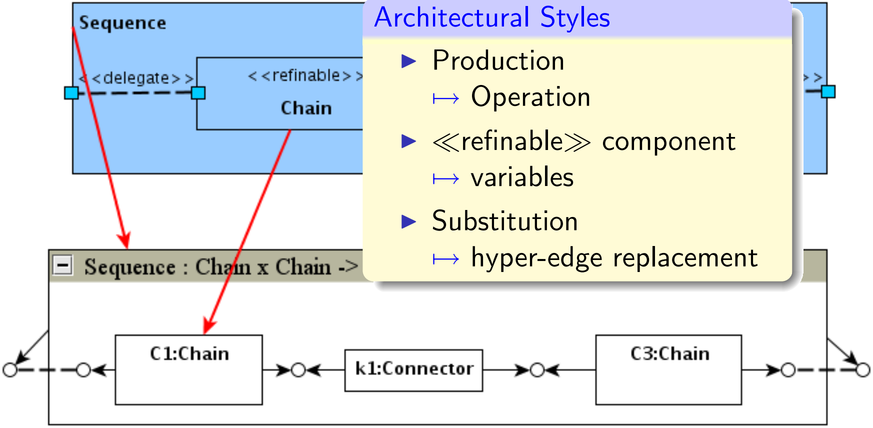
Architectural Styles: Diagrams



Patterns determine the style-conformant compositions:

- ≪refineable≫ component: an architectural type.
- ≪production≫ component: style conformant templates to an architectural type.

Architectural Styles: Underlying Model



Architectural Styles

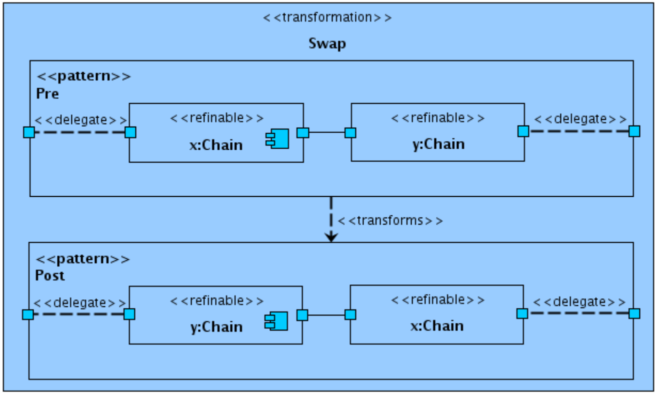
- Production
  $\mapsto$ Operation
- ≪refinable≫ component
  $\mapsto$ variables
- Substitution
  $\mapsto$ hyper-edge replacement

Architectural Styles: Analysis

Does my style T satisfy some given property ϕ?

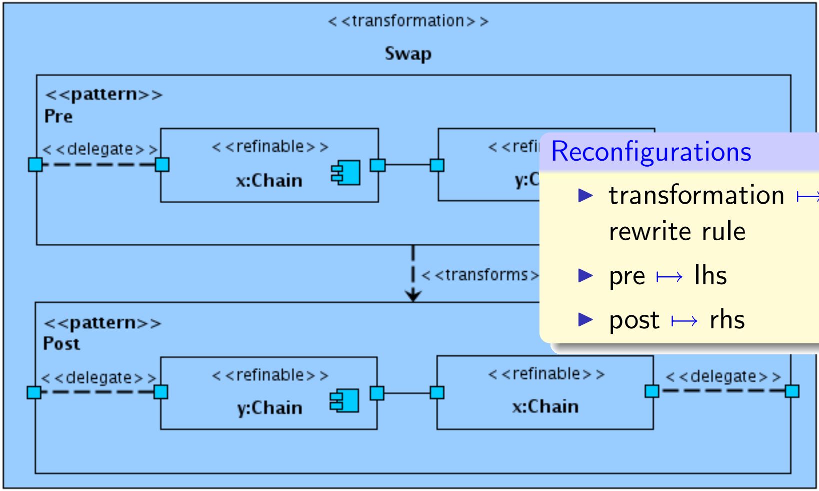
- Property ϕ expressed in some logical language.
- Proof by structural induction: check ϕ on productions for T.
- Example: "***Chain**s are connected*"
  - Check that ϕ holds for production **Single**;
  - Assume ϕ holds and check that it holds for a chain built with **Sequence**.

Reconfigurations: Diagrams



- ≪transformation≫ packages define system reconfigurations;
- ≪pattern≫ diagrams are system templates specifying the system structure before and after the transformation;
- ≪transforms≫ dependencies define the direction of the reconfiguration.

Reconfigurations: Underlying Model



Reconfigurations

- transformation $\mapsto$ rewrite rule
- pre $\mapsto$ lhs
- post $\mapsto$ rhs

Reconfigurations: Analysis

Do all reconfigurations satisfy some linear property?

- Standard exploration algorithms of rewrite engines (e.g. LTL model checking) or semi-automatic verification on rewrite rules.
- Example: *"Filter chains do not grow or decrease"*

CONCLUSION

Concluding Remarks

We have developed an extension of a UML4SOA profile:

- Focus on architectural style-driven reconfiguration of SOA;
- Our formal approach gains a friendly, standard front-end;
- Our UML approach gains formal analysis machinery.

Concluding Remarks

We have developed an extension of a UML4SOA profile:

- Focus on architectural style-driven reconfiguration of SOA;
- Our formal approach gains a friendly, standard front-end;
- Our UML approach gains formal analysis machinery.

Current and future work:

- Integrate the approach in the UML4SOA Tools;
- Concilitate the approach with UML4SOA-R;
- Conciliate with algebraic semantics of MOF.

Credits and Pointers I

Papers

Antonio Bucchiarone, Roberto Bruni, Stefania Gnesi, and Alberto Lluch Lafuente.
Graph-Based Design and Analysis of Dynamic Software Architectures.
In *Concurrency, Graph and Models*, volume 5065 of *LNCS*. Springer Verlag, 2008.

Roberto Bruni, Alberto Lluch Lafuente, Ugo Montanari, and Emilio Tuosto.
Style Based Architectural Reconfigurations.
Bulletin of the European Association for Theoretical Computer Science (EATCS), 94:161–180, 2008.

Nora Koch, Philip Mayer, Reiko Heckel, László Gönczy, and Carlo Montangero.
D1.4a: UML for Service-Oriented Systems.
Specification, SENSORIA Project 016004, 2007.

Daniel Le Métayer.
Describing software architecture styles using graph grammars.
IEEE Transactions on Software Engineering, 24(7):521–533, 1998.

Links

- `http://www.sensoria-ist.eu/`
- `http://www.uml4soa.eu/profile/`
- `http://www.albertolluch.com/adr`

THANKS!