From Architectural to Behavioural Specification of Services

University of Leicester

Laura Bocchi (bocchi@mcs.le.ac.uk), José Luiz Fiadeiro

STUDIORUM UNIVERSITAS FLORENTINA

Alessandro Lapadula, Rosario Pugliese and Francesco Tiezzi

FESCA 2009

Agenda

- Background: SENSORIA, SRML, COWS
- The aim
- The architecture of the implementation
- An example
- Conclusion/future work

Software engineering for SOC

SENSORIA

Software Engineering for Service-Oriented Overlay Computers

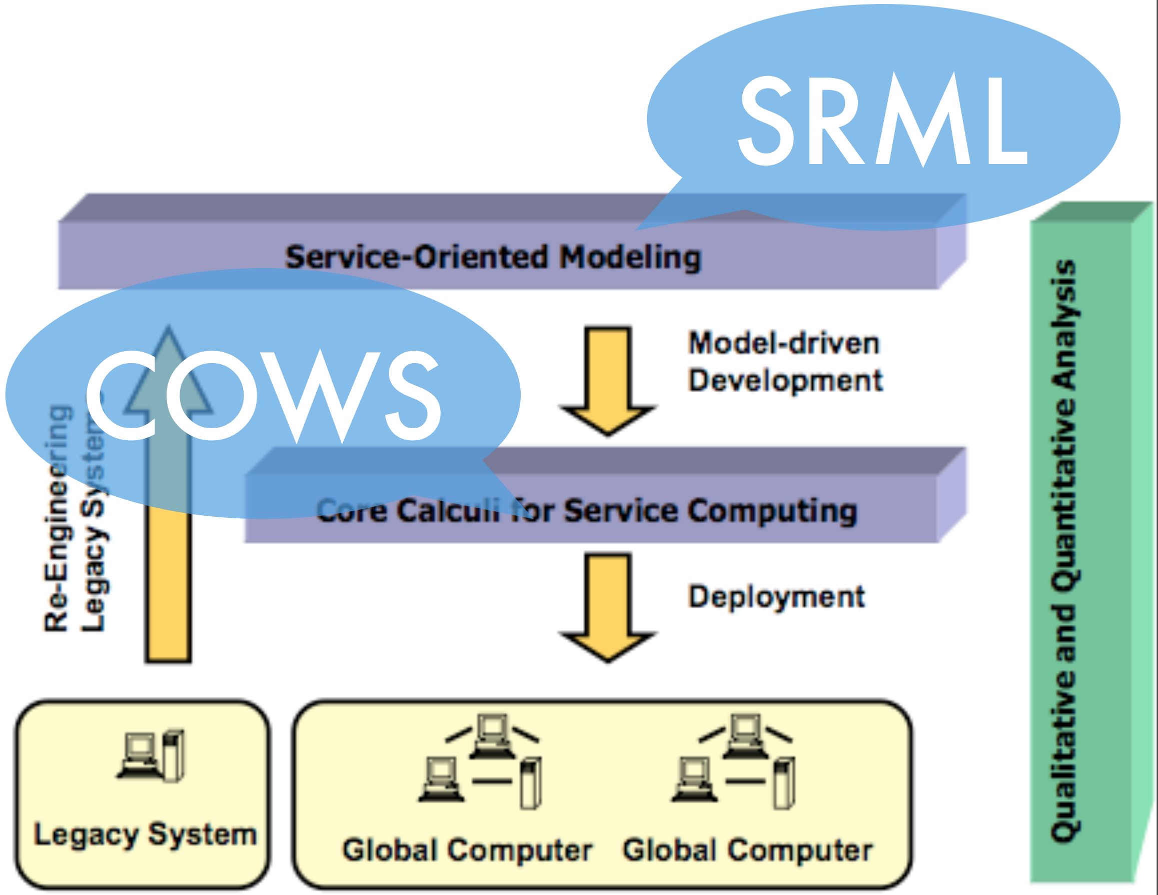
http://www.sensoria-ist.eu/

an IST-FET Integrated Project Sept05–Aug09

- The aim is to develop a novel approach to the **engineering of software systems for service-oriented overlay computers** where foundational theories, techniques and methods are fully integrated in a pragmatic **software engineering approach**

- SOC vs CBD: our view
  - There is no "system" a-priori but an evolving configuration
  - Services add a layer of abstraction over a component infrastructure

- The different languages and formalisms developed in SENSORIA represent each a number of aspects of SOC from different perspectives: none of them aims to be "complete"

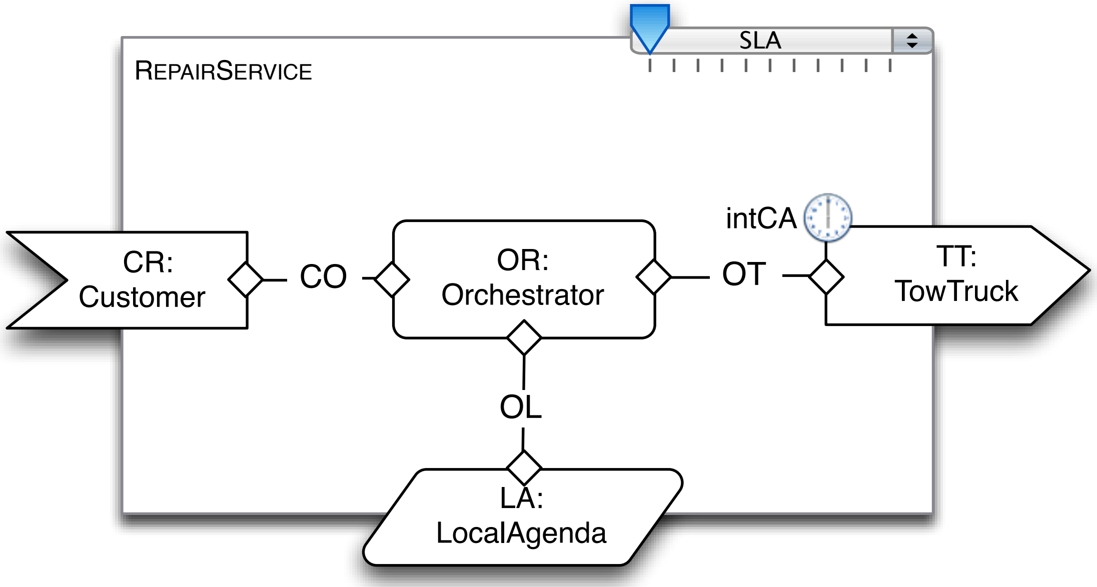

SRML
Service-Oriented Modeling
COWS
Model-driven Development
Re-Engineering Legacy Systems
Core Calculi for Service Computing
Deployment
Legacy System
Global Computer
Global Computer
Qualitative and Quantitative Analysis

SRML&COWS

- SRML: architectural
- SRML is declarative:
  - it supports under-specification
  - it abstracting from how the middleware provides its functionalities

- COWS: behavioural (lower level of abstraction)
- COWS its primitives explicitly model
  - orchestration
  - the functionalities provided by the middleware (e.g., publication, discovery, correlation)

- SRML (overview)



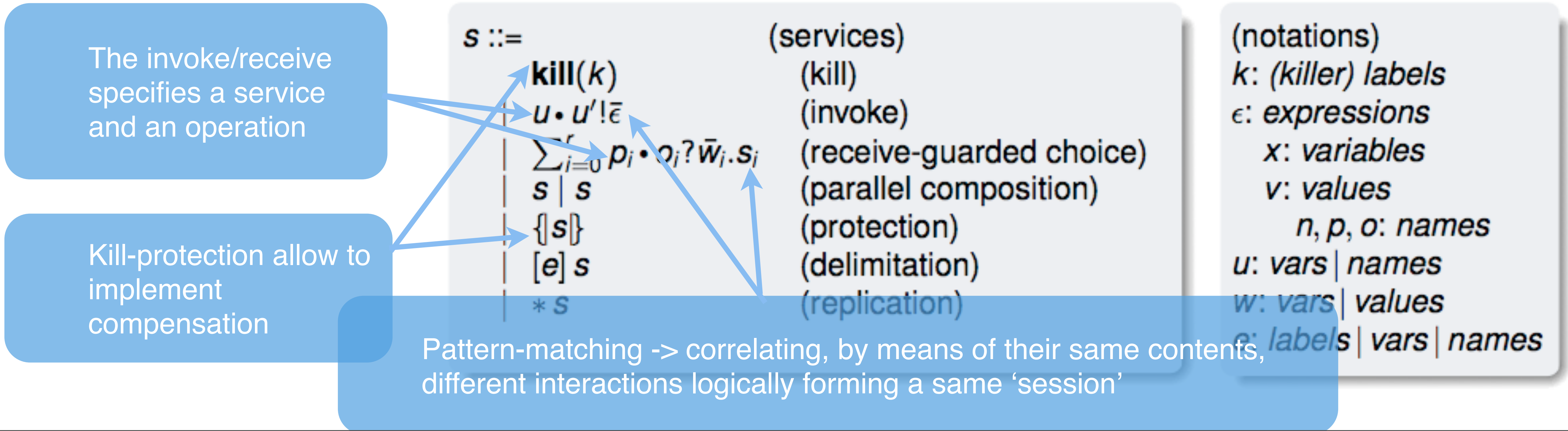
- Module: a number of (different types of) nodes pairwise connected by edges
- Each node *n* has a signature *sign(n)*
- Each node has a (different type of) behavioural interface. All behavioural interfaces are defined in terms of events

SRML&COWS

- SRML: architectural
- SRML is declarative:
  - it supports under-specification
  - it abstracting from how the middleware provides its functionalities

- COWS: behavioural (lower level of abstraction)
- COWS its primitives explicitly model
  - orchestration
  - the functionalities provided by the middleware (e.g., publication, discovery, correlation)

- COWS (overview)



The invoke/receive specifies a service and an operation

Kill-protection allow to implement compensation

Pattern-matching -> correlating, by means of their same contents, different interactions logically forming a same 'session'

s ::= (services)

$\mathbf{kill}(k)$	(kill)
$\mid\ u \bullet u' ! \bar{\epsilon}$	(invoke)
$\mid\ \sum_{i=0}^{r} p_i \bullet o_i ? \bar{w}_i . s_i$	(receive-guarded choice)
$\mid\ s \mid s$	(parallel composition)
$\mid\ \{\mid s \mid\}$	(protection)
$\mid\ [e]\, s$	(delimitation)
$\mid\ * s$	(replication)

(notations)

k: *(killer) labels*

ϵ: *expressions*

x: *variables*

v: *values*

n, p, o: *names*

u: *vars* | *names*

w: *vars* | *values*

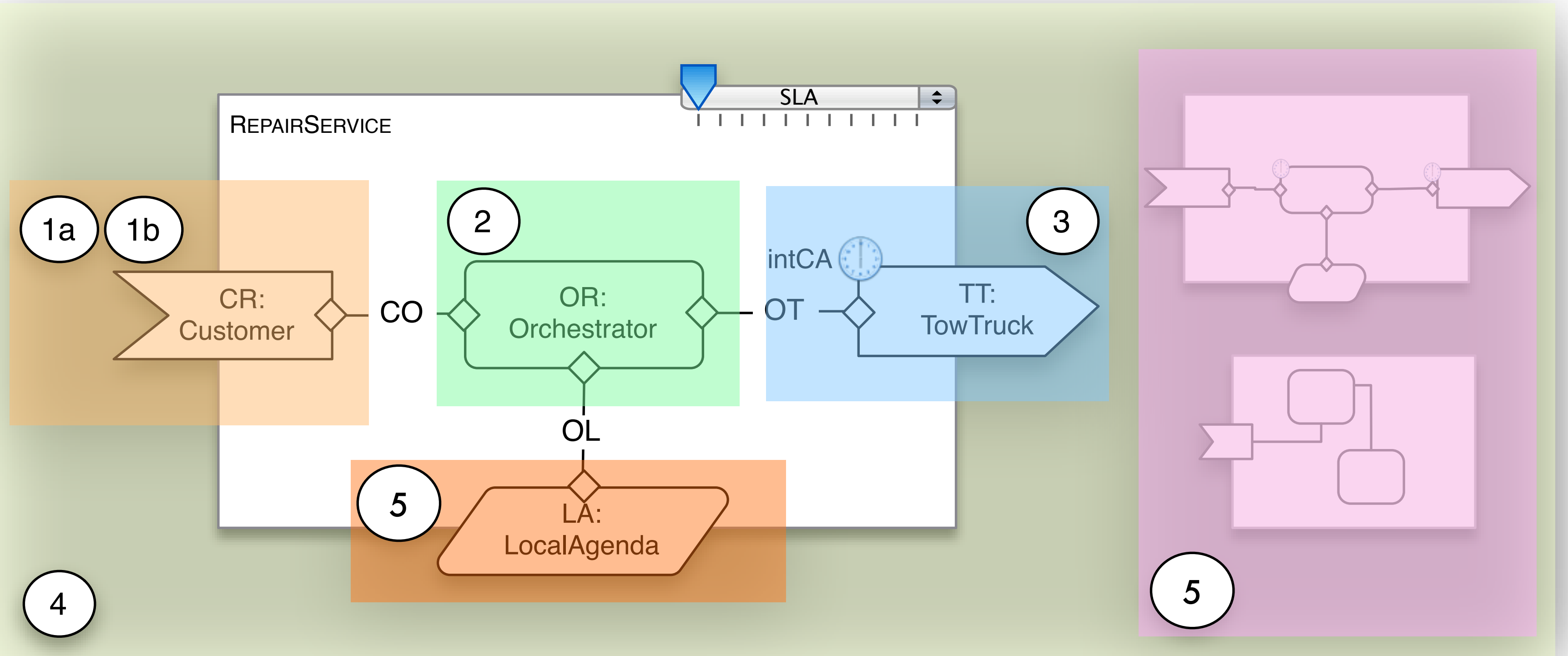
e: *labels* | *vars* | *names*

The Aim

- The **implementation** of those SRML models which are not underspecified in COWS...
  - ...distill minimal set of assumptions made on the middleware
  - ...provides SRML with an operational semantics
  - ...middleware modelled in a way that is is operational but still abstract with respect to implementation issues with actual technologies

Architecture

- SRML



- The implementation of a SRML module into COWS is modular
- COWS

Module$^{(1,2,3)}$ | Middleware$^{(4)}$ | Environment$^{(5)}$

Module$^{(1,2,3)}$=Factory$^{(1a)}$.(InstanceHandler $^{(1b)}$ | Orchestrator $^{(2)}$ | DiscoveryHandler $^{(3)}$)

Creates different instances of a service, each equipped with one instance handler

Implements message correlation to support multiple instances of the same service

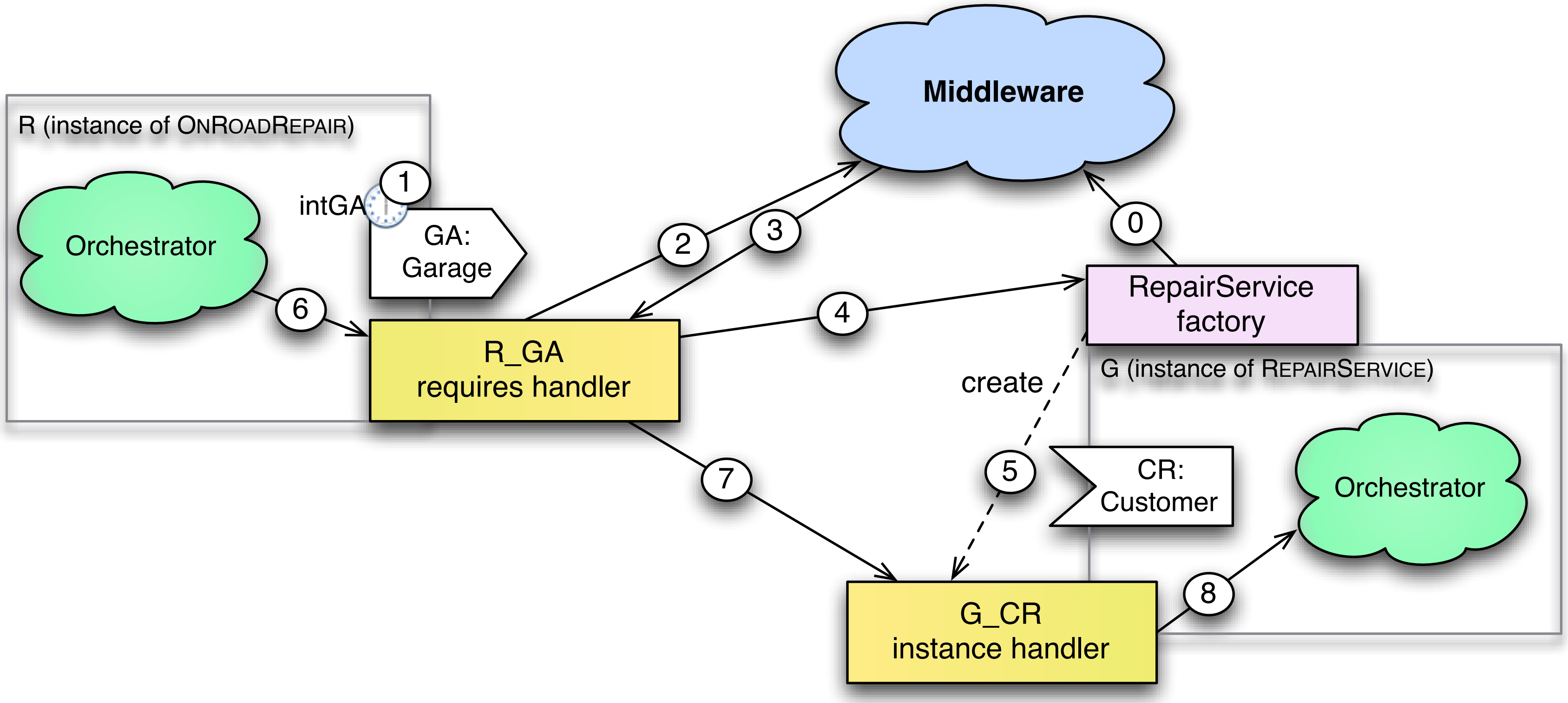
Triggers discovery/binding for each requires-interface and implements message correlation

http://rap.dsi.unifi.it/cows/

http://rap.dsi.unifi.it/cows/papers/SRML2COWS.pdf

Discovery Process

- (1) **intGA** becomes true and triggers the discovery of **GA**
- (2) **R_GA** sends **GA** to **Broker**
- (3) **Broker** returns
  - the id of the best match for **GA**
  - information on the mapping between the names of **GA** and **CR**
- (4) - **R_GA** sends a message to the factory **RepairService** to create a service instance



R_GA

$GA \bullet trigger?\langle id_i \rangle.$ 1

$(\, Broker \bullet disc!\langle OnRoadRepair, id_i, \text{“}Garage\ \mathtt{is}\ \ldots\text{”}, carUserSLAconstraints \rangle$ 2

$\mid [x_p, x_{acceptBooking}]\, OnRoadRepair \bullet GA?\langle id_i, x_p, x_{acceptBooking} \rangle.$ 3

$\quad [id_{ext}]\, (\, x_p \bullet create!\langle OnRoadRepair, id_{ext} \rangle$ 4

$\qquad \mid x_p \bullet bindingInfo!\langle id_{ext}, acceptBookingResp \rangle$

$\qquad \mid *\, [x_{info}]\, GA \bullet acceptBooking?\langle id_i, \text{🔔}, x_{info} \rangle.$

$\qquad\quad (\, x_p \bullet x_{acceptBooking}!\langle id_{ext}, \text{🔔}, x_{info} \rangle$

$\qquad\qquad \mid [x_{servicePrice}]\, OnRoadRepair \bullet acceptBookingResp?\langle id_{ext}, \text{✉}, x_{servicePrice} \rangle.$

$\qquad\qquad\qquad OG_{roleB} \bullet acceptBooking!\langle id_i, \text{✉}, x_{servicePrice} \rangle\,)$

$\qquad \mid \ldots\,)\,)$

G_CR

$*\, [x_{cust}, x_{ext\_id}]\, RepairService \bullet create?\langle x_{cust}, x_{ext\_id} \rangle.$

$\qquad [id_{intra}]\, (\, ProvidesInt \mid RequiresInt \mid Wires \mid Components\,)$

Conclusion/Future Work

- We provided an implementation of SRML modules into COWS
- The aim was to provide SRML primitives with an operational semantics and clarify the assumptions on the middleware
- Focus on dynamic aspects, simplification of some static aspects
- An editor for SRML (Eclipse plugin) has been developed which represents the SRML meta-model as an EMF tree
- Ongoing work - a graphical editor for COWS (based on GMF) with an integrated interpreter
- The automation of the transformation, for example relying on the editors (by means of a transformation between the respective meta-models) would allow SRML models to benefit from the tools for analysis and reasoning made available by COWS:
  - a type system to check confidentiality properties [FSEN07], a temporal logic and a model checker to verify functional properties [FASE08], a static analysis to establish properties of the flow of information between services [SAS08], a stochastic extension to enable quantitative reasoning on service behaviours [ICSOC07], a symbolic representation of the operational semantics [PLACE08], bisimulation-based observational semantics to check interchangeability of services [submitted]

Thank you