

PRELIMINARY PROCEEDINGS

---

AUTOMATED SPECIFICATION  
AND VERIFICATION OF  
WEB SYSTEMS



FIFTH INTERNATIONAL WORKSHOP  
JULY 17, 2009  
CASTLE OF HAGENBERG, AUSTRIA

---

DEMIS BALLIS AND TEMUR KUTSIA (EDS.)



## Preface

This report contains the pre-proceedings of the *5th International Workshop on Automated Specification and Verification of Web Systems* (WWV'09), held at the Research Institute for Symbolic Computation (RISC), Castle of Hagenberg (Austria) on July 17, 2009. The previous editions of the WWV series took place in Siena (2008), Venice (2007), Paphos (2006), and Valencia (2005).

WWV'09 provided a common forum for researchers from the communities of Rule-based programming, Automated Software Engineering, and Web-oriented research, in order to facilitate the cross-fertilization and the advancement of hybrid methods that combine the three areas.

The Program Committee of WWV'09 collected three reviews for each paper and held an electronic discussion during April 2009 which has led to the selection of 10 regular papers. In addition to the selected papers, the scientific program included two invited lectures by François Bry from the Ludwig Maximilian University of Munich (Germany), and Axel Polleres from the National University of Ireland, Galway (Ireland). We would like to thank them for having accepted our invitation.

We would also like to thank all the members of the Program Committee and all the referees for their careful work in the review and selection process. Many thanks to all authors who submitted papers and to all conference participants. We gratefully acknowledge all the institutions and corporations who have supported this event.

Castle of Hagenberg  
July 2009

Demis Ballis and Temur Kutsia  
WWV'09 Editors



# Workshop Organization

## Program Committee

María Alpuente	Technical University of Valencia, Spain
Demis Ballis	University of Udine, Italy
Wlodzimierz Drabent	IDA, Linköping University, Sweden
	IPI PAN, Polish Academy of Sciences, Poland
Santiago Escobar	Technical University of Valencia, Spain
Moreno Falaschi	University of Siena, Italy
Mário Florido	University of Porto, Portugal
Temur Kutsia	Johannes Kepler University Linz, Austria
Massimo Marchiori	University of Padova, Italy
Mircea Marin	University of Tsukuba, Japan
Catherine Meadows	Naval Research Laboratory, USA
Rosario Pugliese	University of Florence, Italy
I.V. Ramakrishnan	Stony Brook University, USA
Antonio Vallecillo	University of Malaga, Spain

## Local Organization

Temur Kutsia  
Wolfgang Schreiner

## External Reviewers

Faisal Ahmed	Mauricio Alba-Castro	Jose Almeida	Michele Baggi
Federico Banti	Artur Boronat	Linda Brodo	Nestor Catano
Jorge Coelho	Asiful Islam	Daniel Romero	Francesco Tiezzi
Alicia Villanueva			

## Sponsoring Institutions

- Land Oberösterreich
- Johannes Kepler University Linz
- Bundesministerium für Wissenschaft und Forschung  
(Austrian Federal Ministry of Science and Research)



## Table of Contents

Linked Broken Data? ( <i>invited talk</i> ) . . . . .	1
<i>Axel Polleres</i>	
Social Media in Sciences ( <i>invited talk</i> ) . . . . .	3
<i>François Bry</i>	
Certified Web Services in Ynot . . . . .	5
<i>Ryan Wisnesky, Gregory Malecha, Greg Morrisett</i>	
Towards a Framework for the Verification of UML models of services . . . .	21
<i>Federico Banti, Francesco Tiezzi, Rosario Pugliese</i>	
Analyzing a Proxy Cache Server Performance Model with the Probabilistic Model Checker PRISM . . . . .	37
<i>Tamas Berczes, Gabor Guta, Gabor Kusper, Wolfgang Schreiner, Janos Sztrik</i>	
Verification of Web Content: A Case Study on Technical Documentation .	53
<i>Christian Schönberg, Mirjana Jaksic, Franz Weitzl, Burkhard Freitag</i>	
A Query Language for OWL based on Logic Programming . . . . .	69
<i>Jesus M. Almendros-Jimenez</i>	
Obtaining accessible RIA UIs by combining RUX-Method and SAW . . . . .	85
<i>Marino Linaje, Adolfo Lozano-Tello, Juan Carlos Preciado, Fernando Sanchez-Figueroa, Roberto Rodríguez</i>	
Automatic Functional and Structural Test Case Generation for Web Applications based on Agile Frameworks . . . . .	99
<i>Boni García, Juan C. Dueñas, Hugo A. Parada G.</i>	
Benchmarking and improving the accessibility of Norwegian municipality web sites . . . . .	115
<i>Morten Goodwin Olsen, Annika Nietzio, Mikael Snaprud, Frank Fardal</i>	
A Rule-based Approach for Semantic Consistency Management in Web Information Systems Development . . . . .	129
<i>Francisco J. Lucas, Fernando Molina, Ambrosio Toval</i>	
Slicing microformats for information retrieval . . . . .	145
<i>J. Guadalupe Ramos, Josep Silva, Gustavo Arroyo, Juan Carlos Solorio</i>	





# Linked Broken Data?

Axel Polleres

Digital Enterprise Research Institute, National University of Ireland, Galway\*  
axel.polleres@deri.org,  
Home page: <http://www.polleres.net/>

**Abstract.** The Semantic Web idea is about to grow up. By efforts such as the Linked Open Data initiative, resources like Wikipedia, movie & music databases, news archives, online citation indexes, social networks but also product catalogues and reviews are becoming available in structured form as RDF using common ontologies and we finally find ourselves at the edge of a Web of Data becoming reality. Emerging standards such as OWL, RIF and SPARQL shall allow us to reason and ask structured queries on this data, but – as we will see – the novel Web of Data is still brittle. In this talk we will report about our experiences when reasoning about Web data at large scale. In the first part of this talk, we will outline the ideas of linked data, and briefly introduce RDF, OWL and SPARQL by examples taken from the Web. We will then report about experiences when applying rule-based OWL reasoning on Web data: sound and complete reasoning techniques soon reach their limits when fired at real Web data and we have to cut back to still arrive at reasonable inferences. Our experiments also have revealed common mistakes when publishing linked data for which we provide a preliminary validation service. We will wrap up our findings with a discussion of open challenges.

## References

1. Aidan Hogan, Andreas Harth, and Axel Polleres. Scalable authoritative owl reasoning for the web. *International Journal on Semantic Web and Information Systems*, 5(2), 2009.
2. Renaud Delbru, Axel Polleres, Giovanni Tummarello, and Stefan Decker. Context dependent reasoning for semantic documents in Sindice. In *Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2008)*, Karlsruhe, Germany, October 2008.

---

\* The presented results are based on and extending joint work with Aidan Hogan, Andreas Harth, Renaud Delbru, Giovanni Tummarello and Stefan Decker [1, 2]. This work was funded by Science Foundation Ireland (SFI) under the project SFI/08/CE/I1380 (Lion-2).



# Social Media in Sciences

François Bry

Institute for Informatics, Ludwig-Maximilian University of Munich, Germany

Email: [bry@lmu.de](mailto:bry@lmu.de),

Home page: <http://www.pms.ifi.lmu.de/mitarbeiter/bry/>

**Abstract.** Social media such as digital networks, wikis, social tagging platforms and blogs are now firmly established in our everyday life. Social media have begun to enter the workplace of many, in particular of scientists. This presentation first casts a glance at today's widespread social media, then look at these media from various angles, successively addressing the social vision they are underpinned with, their characteristics, and analyses of the social media phenomenon. In a second part, this presentation addresses the emerging use of social media in sciences. First, this bubbling and fast changing field is introduced by a few characteristic media. Then, personal, admittedly subjective, views are proposed on the kind of social media susceptible of becoming popular in sciences. Based on these views, research issues related to such media are suggested. Finally, research activities the speaker is involved in related to social media for sciences and technology are presented.



# Certified Web Services in Ynot

Ryan Wisnesky, Gregory Malecha, and Greg Morrisett

Harvard University  
{ryan, gmalecha, greg}@cs.harvard.edu

**Abstract.** In this paper we demonstrate that it is possible to implement certified web systems in a way not much different from writing Standard ML or Haskell code, including use of imperative features like pointers, files, and socket I/O. We present a web-based course gradebook application developed with Ynot, a Coq library for certified imperative programming. We add a dialog-based I/O system to Ynot, and we extend Ynot’s underlying Hoare logic with event traces to reason about I/O behavior. Expressive abstractions allow the modular certification of both high level specifications like privacy guarantees and low level properties like memory safety and correct parsing.

## 1 Introduction

In this paper we demonstrate that it is possible to implement certified web systems in a way not much different from writing Standard ML or Haskell code, including use of imperative features like pointers, files, and socket I/O. We present a web-based course gradebook application developed with Ynot [18], a Coq [1] library for certified, general-purpose, imperative programming.

Our specification of application behavior, imperative application implementation, and certification that the implementation meets the specification are all written in the dependently-typed, higher-order  $\lambda$ -calculus of inductive constructions (CIC) [1]. Ynot links user code with an imperative extension to the Coq proof assistant, resulting in an executable web server. We add a dialog-based [12] I/O system to Ynot, and we extend Ynot’s underlying Hoare logic with event traces [3] to reason about I/O behavior. Expressive abstractions allow the modular certification of both high level specifications like privacy guarantees and low level properties like memory safety and correct parsing. The proof that the application meets its specification has been developed semi-automatically and interactively during development, imposes no runtime overhead, and can be verified in minutes by a several-hundred-line CIC typechecker.

We give an overview of the course gradebook server in Section 2 and describe Ynot in Section 3. Our I/O library is described in Section 4. The certified imperative implementation is discussed in Section 5, followed by a comparison to related tools in Section 6. We conclude with lessons learned and thoughts on future work. The source code is included in the Ynot distribution at [ynot.cs.harvard.edu](http://ynot.cs.harvard.edu).

## 2 Overview

Our goal is to build a simple, web-based course gradebook that allows students, teaching assistants, and professors the ability to read, edit, and statistically aggregate grades in a privacy-respecting way. We use a traditional three-tiered web application architecture with role-based privacy, a persistent backend data store, an application logic layer, and a presentation component [19].

We specify the store using a purely functional implementation of a minimal subset of SQL, including basic select, project, update, insert, and delete commands. We have implemented an imperative store using a pointer-based data structure, but this detail is isolated from the rest of the system by higher-order separation logic [20]. External databases may also be used this way.

The application logic specifies the behavior of the gradebook using high-level domain-specific concepts like grades, assignments, and sections. For instance, the specification states that students should not be allowed to see each other's grades. Imperative implementations are proven correct with respect to this model.

To users, the gradebook application appears as a regular HTML-based website, with commands sent to the application using HTTP. The application server parses HTTP requests by compiling input PEG grammars [8] to packrat parsing computations in a certified way [15]. An executable webserver is generated by extraction from Coq to OCaml [1].

## 3 Ynot

In this section we introduce Ynot by means of increasingly comprehensive examples. We begin with `helloworld`:

```
Definition helloworld : Cmd empty (fun _ : unit => empty) :=  
  printStringLn "Hello World".
```

Imperative computations (also called commands) like `printStringLn` have `Cmd` types; `Cmd` is analogous to the `IO` monad in Haskell and is indexed by pre- and post-conditions as in Hoare Type Theory [17]. Ynot allows non-terminating recursion, and post-conditions are partial correctness assertions. Hence `helloworld` is a computation whose pre-condition is that the heap is empty, and whose post-condition is that the computation, if it doesn't diverge, returns a unit and ensures that the heap is empty.

The heap is accessed through the traditional `new`, `read`, `write`, `free` commands, which we reason about using separation logic [20]. Their types are:

```
Definition SepNew (T : Type) (v : T) : Cmd empty (fun p => p --> v).
```

```
Definition SepFree (T : Type) (p : ptr)  
  : Cmd (Exists v :@ T, p --> v) (fun _ : unit => empty).
```

```
(* SepRead is also written ! *)
```

```

Definition SepRead (T : Type) (p : ptr) (P : T -> hprop)
  : Cmd (Exists v :@ T, p --> v * P v) (fun v => p --> v * P v).

(* SepWrite is also written ::= *)
Definition SepWrite (T T' : Type) (p : ptr) (v : T')
  : Cmd (Exists v' :@ T, p --> v') (fun _ : unit => p --> v).

```

Pre- and post-conditions are predicates over the heap (**hprops**). A `:@` indicates the type of an existential quantifier. The `p --> v` represents the **hprop** that `p` points to `v`. For example, when **SepNew** is run in the empty heap with argument `v`, it returns a pointer<sup>1</sup> to `v`. **SepFree** is the inverse: it takes a valid pointer and frees it, hence the post-condition is the empty heap. Note that **SepFree**'s type does not mean that the entire heap is empty, only that the portion of the heap referred to by the pre-condition is empty – this is characteristic of the small-footprint approach of separation logic. Pointers in Ynot are not explicitly typed, so the **SepWrite** function allows changing the type of the value pointed to by `P`. The `*` is separation logic conjunction, indicating that the heap can be split in to two disjoint portions that satisfy each conjunct. **SepRead**'s type indicates that to read `p`, `p` must point to some `v`; the additional parameter `P` can be used to dependently describe the heap around `p`, and is useful for proof automation but not strictly required.

As in Haskell, commands are sequenced through monadic binding. Intuitively, binding two computations `c1` and `c2` means running `c1` and then running `c2` using the result of `c1` as input: we write this as `v <- c1; c2 v`, and as `c1 ;; v2` when `c1`'s output is ignored. Binding requires us to prove that the pre-condition of `c2` is a logical consequence of the post-condition of `c1`. In general, we write imperative code first and then prove the correctness of sequencing afterward, using a Coq tactic called **refine**. For example, the following program swaps the values of two pointers:

```

Definition swap (p1 p2 : ptr) (v1 : [nat]) (v2 : [nat]) :
  Cmd (
    v1 ~~ v2 ~~ p1 --> v1 * p2 --> v2)
    (fun _ : unit => v1 ~~ v2 ~~ p1 --> v2 * p2 --> v1).
refine (fun p1 p2 =>
  v1 <- ! p1 <@> (v2 ~~ p2 --> v2);
  v2 <- ! p2 <@> _ ;
  p1 ::= v2 ;; {{ p2 ::= v1 }});
sep inst auto.
Qed.

```

The type of **swap** expresses that **swap** takes as arguments two pointers and two computationally irrelevant natural numbers such that the first pointer points to the value of the first and analogously for the second. If **swap** terminates, then the first pointer will point to the second value and the second will point to the first value. The `[-]` marks a parameter as computationally irrelevant. Such parameters only serve specification purposes and do not affect runtime

---

<sup>1</sup> Ynot does not allow unrestricted pointer arithmetic, so pointers are essentially references/locations.

behavior; they are erased during compilation. Irrelevant values must be explicitly unpacked: `v1` has type `[nat]`, but inside the `v1~~` it can be treated as a `nat`.

The `swap` function itself is similar to a typical pointer-swapping function but includes extra information to help us prove partial correctness. `refine` generates proof obligations, which we here discharge using Ynot's built in separation logic tactic, `sep`. Within the `refine`, `<@>` is a use of separation logic's frame rule, which allows us to describe the portion of the heap that a computation doesn't use. In this example, for instance, we need to know that `p2` points to `v2` before and after `p1` is read. This fact can actually be inferred automatically, but we write it out here for sake of explanation. In the following line, the `_` indicates that the framing condition should be inferred. Finally, the `{{-}}` indicates that the type of the final write may need its pre-condition strengthened and its post-condition weakened to match the overall type of `swap`.

The memory correctness properties of our implementation, such as absence of null pointer dereferences and memory leaks, are statically guaranteed at compile-time. For example, consider the following erroneous program:

```
Definition leak : Cmd empty (fun _ : unit => empty).
  refine (v <- SepNew 1 ; {{ Return tt }}).
```

Because the heap contains 1 after the call to `New` but the return type of `leak` states that the heap should be empty, `refine` generates a false obligation:

```
v --> 1 ==> empty
```

We achieve modularity in Ynot by defining abstract interfaces for imperative components so that many implementations can be used. Consider the following interface and implementation of a simple counter:

```
Module Type Counter.
  Parameter T : Type. (* type of implementation *)
  Definition M := nat. (* type of logical model *)
  Parameter rep : T -> M -> hprop. (* heap representation *)

  Parameter inc : forall (t : T) (m : [M]),
    Cmd (m ~~ rep t m) (fun _ : unit => m ~~ rep t (m + 1)).
End Counter.

Module CounterImpl : Counter.
  Definition T := ptr.
  Definition rep (t : T) (m : M) := t --> m.
  ...
End CounterImpl.
```

`T` is the type of the imperative implementation, which corresponds to a pointer for this implementation. `M` is the logical model for the data structure, in this case a natural number which is the current value in the counter. The `rep` function relates, through an `hprop`, the state of the imperative implementation to logical model. The `forall` keyword indicates a dependent function type: `inc`'s post-condition depends on `m`. The module type hides everything but the logical model, providing an abstraction barrier for users of the module.



## 4 Files, Sockets, and Traces

We have extended Ynot with an axiomatic networking and file I/O library based on the OCaml Unix library. Just as we record the effect of memory operations using separation logic, we record the effects of I/O actions using a trace [3]:

```
Axiom Action : Set.  
Definition Trace := list Action.  
Axiom traced : Trace -> hprop.
```

Our type of `Actions` is open [14], allowing library users to define additional I/O events. Traces are defined as lists for convenience, and we will only be reasoning about finite trace fragments.

We include file and socket operations such as `read`, `write`, `accept`, etc.. The UDP send operation, for instance, is exposed as `(List cons` is written as `::` in Coq):

```
Axiom SocketAddr : Set.  
Axiom Sent : SocketAddr -> SocketAddr -> list ascii -> Action.  
  
Axiom send : forall (local remote : SocketAddr)  
  (s : list ascii) (tr : [Trace]),  
  Cmd (tr ~~ traced tr)  
    (fun _ : unit => tr ~~ traced (Sent local remote s :: tr)).
```

We do not formally verify our OCaml code, which for the most part delegates to Unix functions. For instance, we do not verify the implementation of the TCP state machine, although it is possible to do so [2].

Traces can be reasoned about using temporal logics [6]; however, for simplicity, we reason about them directly using inductive Coq definitions. For instance, the following Coq datatype specifies correct, properly echoed, traces of an echo server:

```
Inductive echoes (local : SocketAddr) : Trace -> Prop :=  
  | NilEchoes : echoes local nil  
  | ConsEchoes : forall remote s past, echoes local past ->  
    echoes local (Sent local remote s :: Recd local remote s :: past).
```

Here each `|` indicates a data constructor. This definition expresses that the empty trace is allowable (`NilEchoes`), and that if some trace `past` is allowable, then additionally echoing back a single request is also allowable (`ConsEchoes`). The `|` symbol is also used in `match` expressions that eliminate inductive types.

### 4.1 Certified Application Servers

Many web systems, including our gradebook server, can be structured as computations that an application server executes repeatedly. Such web applications can be programmed using event loops in the style of dialogs [12], and our I/O library contains support for proving such systems correct with respect to a trace [3]

semantics. At a minimum, an application iteration is defined by an invariant-preserving Ynot function that is runnable in the initial world of an empty heap and empty trace. For instance, the type of an echo application is:

```
Definition echo_iter_t local := forall (tr : [Trace]),
  Cmd (tr ~~ traced tr * [echoes local tr])
    (fun _ : unit => tr ~~ Exists r :@ Trace,
      traced (r ++ tr) * [echoes local (r ++ tr)]).
```

The `[]` notation is overloaded here to indicate “pure” propositions which do not mention the heap. List concatenation is written `++`. A computation of this type, when repeated any number of times, beginning in the initial world, always generates a trace that is in `echoes`. An example echo implementation that conforms to the above specification is:

```
Definition echo (local : SockAddr) : echo_iter_t local.
unfold echo_iter_t; refine ( fun local tr =>
  x <- recv local tr <@> _ ;
  {{ send local (fst x) (snd x) (tr ~~~
    (Recd local (fst x) (snd x) :: tr)) <@> _ }} );
rsep fail auto.
Qed.
```

We have written out the intermediate state of the trace history (using an irrelevant value unpacking operation `~~~`), but such states can often be inferred.

We have implemented a number of UDP, TCP, and SSL application servers. In each case their types ensure that they only run applications that preserve some notion of partial correctness. The simplest, the `forever` server, repeats a given computation forever. The implementation of `forever` is half a dozen lines, does not require a single line of manual proof, and includes the post-condition that the server never halts.

## 5 The Application

In this section we describe the gradebook application specification, our imperative implementation of it, and the proof that the implementation meets the specification. We begin with the purely functional specification of the gradebook itself (Section 5.1). We then describe the entire deployed application server starting from the backend and working toward the user. We start with the data store (Section 5.2) which provides the data manipulation operations we use in our imperative implementation (Section 5.3). From there, we show how the application can be deployed using our application server (Section 5.4). We conclude with an explanation of the frontend (Section 5.5) in which we focus on parsing user requests. It is helpful to keep in mind that every imperative component must be related to a purely functional model.

## 5.1 Application Logic

In this section we define the specification of our application. We begin by defining the configuration of a course:

```

Definition Section      := nat.
Definition Password     := nat.
Definition Grade        := nat.
Definition Assignment   := nat.
Record Config : Set := mkCf {
  students, tas, profs : list ID;
  sections : list (ID * Section);
  hashes   : list (ID * Password);
  totals   : list Grade
}.

```

We are using natural numbers for our basic types, but abstract types can also be used. Configurations are specified to have a number of properties; for example, all students, teaching assistants and professors must have a password. These properties are given by a Coq definition:

```

Definition correct_cfg (cfg : Config) := forall id,
  (In id (students cfg) /\ In id (tas cfg) /\ In id (profs cfg) ->
    exists hash, lookup id (hashes cfg) = Some hash) /\ ...

```

The actual grades are modeled by a `list (ID * list Grade)`. Like with the configuration, we define a predicate `gb_inv` to ensure the integrity of the grade data. Among other things, this specifies that grade lists must always be the length of the totals list given in the configuration, each grade must be less than the associated maximum permissible, and each student must have an entry.

The gradebook application manages a single course by processing user commands, updating the grades if necessary, and returning a response. The available commands are given by a Coq datatype:

```

Inductive Command : Set :=
| SetGrade : ID -> PassHash -> ID -> Assignment -> Grade -> Command
| GetGrade : ID -> PassHash -> ID -> Assignment -> Command
| Average  : ID -> PassHash -> Assignment -> Command.

```

The meaning of the commands is given by a pure Coq function `mutate` that maps a `Command`, `Config`, and `list (ID * list Grade)` to a new `list (ID * list Grade)` and a response:

```

Inductive Response : Set :=
| ERR_NOTPRIVATE : Response | ERR_BADGRADE : Response
| ERR_NOINV      : Response | OK      : Response | RET : Grade -> Response.

```

There are numerous ways to define the desired gradebook behavior, but using a total function makes the application easy to deploy with our application server.

Privacy is enforced using simple role based access control; `private` is a predicate that defines when commands are privacy respecting, and non-private queries are specified to return `ERR_NOTPRIVATE`:

```

Definition isProf (cfg: Config) (id: ID) (pass: Password) :=
  In id (profs cfg) /\ lookup id (hashes cfg) = Some pass.
...

Definition private (cfg : Config) (cmd : Command) : Prop :=
  match cmd with
  | SetGrade id pass x _ => isProf cfg id pass /\ taFor cfg id pass x
  | GetGrade id pass x _ => isProf cfg id pass /\ taFor cfg id pass x
                        /\ (id = x /\ isStudent cfg id pass)
  | Average id pass _    => isProf cfg id pass /\ isStudent cfg id pass
                        /\ isTa cfg id pass
  end.

```

We have proved a number of theorems about this specification, like that `mutate` preserves `gb_inv` and will not return `ERR_NOINV` when `gb_inv` holds. To help make the proofs more tractable, we implemented a number of automated proof search tactics tailored to this model.

## 5.2 Data Store

The backend data store of the gradebook server is a simplified relational database. We first give a functional specification of the store, and then prove that our imperative implementation meets this specification. Logically, a **Store** is modeled by a list of **Tuple n** defined by the following Coq datatype:

```

Fixpoint Tuple (n: nat) : Type :=
  match n with
  | 0 => unit
  | S n' => (nat * Tuple n')
  end.

Definition Table n : Type := list (Tuple n).

```

For simplicity we are storing only natural numbers, and we specify only a small subset of the functionality of SQL, including select, update, project, and delete. For instance, selection is modeled logically by:

```

Definition WHERE := Tuple n -> bool. (* ‘where’ clause *)

Fixpoint select (wh : WHERE) (rows : Table) : Table :=
  match rows with
  | nil => nil
  | a :: r => if wh a then a :: select wh r else select wh r
  end.

```

Our purely functional model has expected properties, like:

```

Theorem select_just : forall tbl tbl' wh, select wh tbl = tbl' ->
  (forall tpl, In tpl tbl' -> wh tpl = true /\ In tpl tbl).

Theorem select_all : forall tbl tbl' wh, select wh tbl = tbl' ->
  (forall tpl, In tpl tbl -> wh tpl = true -> In tpl tbl').

```

Persistence is reflected in the store interface by the simple requirement that serialization and deserialization be inverses:

```

Parameter serial    : Table n -> string.
Parameter deserial  : string  -> option (Table n).
Parameter serial_deserial : forall (tbl : Table n),
  deserial (serial tbl) = Some tbl.

Parameter serialize : forall (r : t) (m : [Table n]),
Cmd (m ~~ rep r m) (fun str:string => m ~~ rep r m * [str = serial m]).
Parameter deserialize : forall (r : t) (s : string),
  Cmd (rep r nil)
    (fun m : option [Table n] =>
      match m with
      | None => rep r nil * [None = deserial s]
      | Some m => m ~~ rep r m * [Some m = deserial s]
      end).

```

We have implemented the store using an abstract linked-list. The linked-list has a several imperative implementations, including one using pointers to list segments. We found the linked-list's effectful fold operation particularly useful.

### 5.3 Certified Implementation

Based on the specification given in Section 5.1, a certified implementation of our gradebook meets the following interface:

```

Module Type GradeBookAppImpl.
  Parameter T : Set.
  Parameter rep : T -> (Config * list (ID * (list Grade))) -> hprop.

  Parameter exec : forall (t : T) (cmd : Command)
    (m : [Config * list (ID * (list Grade))]),
    Cmd (m ~~ rep t m * [gb_inv (snd m) (fst m) = true])
      (fun r : Response => m ~~ [r = fst (mutate cmd m)] *
        rep t (snd (mutate cmd m)) * [gb_inv (snd m) (fst m) = true]).
End GradeBookAppImpl.

```

Note that the `exec` computation is invariant preserving, can only be run when the invariant is satisfied, and faithfully models `mutate`. For convenience, we keep the course configuration in memory at runtime, and we parameterize our implementation by an abstract backend store:

```

Module GradeBookAppStoreImpl (s : Store) : GradeBookAppImpl.
  Definition T := (Config * s.T).

```

In trying to write `rep`, we immediately encounter an impedance mismatch between our logical gradebook model (based on `list (ID * list Grade)`) and the table based model of the store (based on `Tuples`). Following the 3-tier web application model, we define an object-relational mapping [13] between the domain-specific objects of students, grades, etc., and the relational store:

```

Module GradesTableMapping.
  Fixpoint Tuple2List' n : Tuple n -> list Grade :=
    match n as n return Tuple n -> list Grade with
    | 0 => fun _ => nil
    | S n => fun x => (fst x) :: (Tuple2List' n (snd x))
    end.
  Definition Tuple2List n (x : Tuple (S n)) :=
    match x with
    | (id, gr) => (id, Tuple2List' n gr)
    end.
  Fixpoint Table2List n (x : Table (S n)) : list (ID * list Grade) :=
    match x with
    | nil => nil
    | a :: b => Tuple2List n a :: Table2List n b
    end.
End GradesTableMapping.

```

The list to table direction is similar. Other data models, such as with three-tuples (id, assignment, grade), require different mappings, but regardless of the choice of data model and mapping we must prove that the mapping is an isomorphism from the logical model to the data model:

```

Theorem TblMTbl_id : forall l c, store_inv1 l c = true ->
  Table2List (length (totals c))
  (List2Table l (length (totals c))) = l.

```

Isomorphism is actually an overly strong requirement, but it helps simplify reasoning. With the mapping to the data model done, we can define the concrete imperative representation:

```

Definition rep (cfg, t) (cfg', gb) :=
  [cfg = cfg'] * s.rep t (List2Table gb)

```

The imperative implementation consists of a runtime configuration `cfg` and a handle to an imperative store `t`, which `rep` relates to the logical gradebook model. `rep` states that the runtime configuration (`cfg`) is identical to the logical model's configuration (`cfg'`), and that the imperative gradebook's state (`t`) is isomorphic to the logical model's (`List2Table gb`). The complete imperative implementation consists of hundreds of lines of code, proofs, and tactics, so we can only give highlights here. The implementation of retrieving a grade, omitting some definitions, is:

```

Definition F_get user pass id assign m t :
  Cmd (m ~~ rep t m * [store_inv (snd m) (fst m) = true] *
    [private (fst t) (GetGrade user pass id assign) = true])
    (fun r : Response => m ~~ [store_inv (snd m) (fst m) = true] *
      [r = fst (mutate (GetGrade user pass id assign) m)] *
      rep t (snd (mutate (GetGrade user pass id assign) m))).
refine (fun user pass id assign m t =>

```

```

res <- s.select (snd t) (get_query id (fst t))
      (m ~~~ List2Table (snd m)
        (length (totals (fst t))) ) <@> _ ;
match nthget assign res as R
return nthget assign res = R -> _ with
| None => fun pf => {{ !!! }}
| Some w => fun pf =>
  match w as w' return w = w' -> _ with
  | None => fun pf2 => {{ Return ERR_BADGRADE }}
  | Some w' => fun pf2 =>
    {{ Return (RET w')
      <@> (m ~~ [fst t = fst m] *
        [store_inv (snd m) (fst t) = true] *
        [private (fst t) (GetGrade user pass id assign) = true]) }}
    end (refl_equal _)
  end (refl_equal _) ).

```

The intuition here is that we first run a `get_query` over the store `s`, which results in a table `res`. Because the gradebook invariant holds, `res` contains a single tuple of the requested student's grades. `nthget` returns `None` if the input table is empty, so we mark this branch as impossible (`!!!`). We then project out the desired grade, returning an error if there is no such requested assignment. The proof script for this function is almost completely automated and consists almost entirely of appeals to `sep` and uses of purely logical theorems about the application model. For instance, a typical proof about the specification is:

```

Theorem GetGrade_private_valid : forall (T : Type) x (t : Config * T)
  user pass id assign, fst t = fst x
-> store_inv (snd x) (fst x) = true
-> private (fst t) (GetGrade user pass id assign) = true
-> nthget assign (select (get_query id (fst t))
  (List2Table (snd x) (length (totals (fst t))))) <> None

```

This theorem states that if the `get` command is privacy respecting, then the student has a grade. The other operations are implemented analogously.

## 5.4 Deploying to an Application Server

To deploy our application using our read-parse-execute-prettyprint application server we must implement:

```

Module Type App.
  Parameter Q : Set.                (** type of app's input *)
  Parameter R : Set.                (** type of app's output *)

  Parameter T : Set.                (** type of imperative app *)
  Parameter M : Type.              (** type of logical app model *)
  Parameter rep : T -> M -> hprop. (** app representation invariant *)

```

```

(** the functional model of the application *)
Parameter func  : Q -> M -> (R * M).
Parameter appIO : Q -> M -> (R * M) -> Trace.

(** the app implementation *)
Parameter exec : forall (t : T) (q : Q) (m : [M]) (tr : [Trace]),
  Cmd (tr ~~ m ~~ rep t m * traced tr)
      (fun r : R => tr ~~ m ~~ let m' := snd (func q m) in
        [r = fst (func q m)] *
        rep t m' * traced (appIO q m (r,m') ++ tr)).

```

This interface requires a functional application model (`func`), and allows the application to transparently perform I/O operations by wrapping the desired sequence in the `appIO` trace. The gradebook application only performs I/O on startup and shutdown, and so it meets this interface trivially. (Startup and shutdown are straightforward, so we do not discuss them.) The application server also requires a parser and frontend, which are defined by the following functions and discussed in the following subsection:

```

Parameter grammar : Grammar Q.
Parameter parser   : parser_t grammar.
Parameter printer  : R -> list ascii.
Parameter err      : parse_err_t -> list ascii.

```

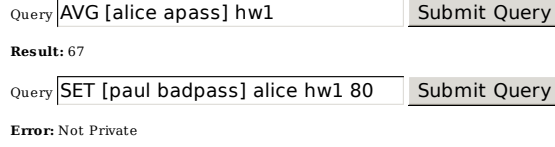
With these definitions in place, we can describe the traces of a correct application implementation, which we do using an inductive datatype in the same way we specified correctness for the echo server (Section 4). Either the input request parsed correctly, and the result was sent to the application for processing and the response returned to the user, or the parse failed and an error was returned.

## 5.5 Frontend

The frontend parses inputs into `Commands` and converts application `Responses` into text. For instance, we have implemented a raw-sockets frontend by using our packrat PEG parser and straightforwardly printing responses. We have also implemented an HTML frontend as an application transformer: given an application, the HTML frontend passes along certain HTTP fields to the application and converts responses to HTML output. Several screen shots of the application running with a minimal skin are given in Figure 1.

The HTTP server uses a packrat PEG parser written in Ynot to parse HTTP requests. The parser is implemented as a certified compiler [15]: given a specification consisting of a PEG grammar and semantic actions, the parser creates an imperative computation that, when run over an arbitrary imperative character stream, returns a result that agrees with the specification. To make the parsing efficient, the packrat algorithm employed by the resultant imperative computation uses a sophisticated caching strategy which is implemented using imperative hashtables. We may also write our own parsers against the parser interface.





**Fig. 1.** Screenshots of the gradebook running in Mozilla Firefox.

## 5.6 Evaluation

Figure 2 describes the breakdown of proofs, specifications, and imperative code in our certified components. Program code is Haskell-ish code that has a direct analog in the executed program (e.g. `F_get`). Specs are model definitions but not proofs (e.g. `gb_inv`). Proofs counts all proofs (e.g. `select_just`) and tactic definitions. Overhead gives the ratio of proofs to program code and the time column indicates proof search and checking time on a 2Ghz Core 2 laptop with 2GB RAM. We have made no attempt to optimize any of these numbers. These totals do not include the base Ynot tactics and data structures that we use, which include an imperative hashtable, stream, and segmented linked list.

	Program	Specs	Proofs	Overhead	Time (m:s)
Packrat PEG Parser	269	184	82	.3	0:55
Store	113	154	99	.88	0:23
Gradebook Application	119	231	564	4.74	0:32
HTTP-SSL-TCP Application Server	223	414	231	1.04	1:21
Other I/O Library	90	76	90	1	0:05

**Fig. 2.** Numbers of lines of different kinds of code in the imperative components

The ratios of overhead vary, but the application stands out as having the largest proof burden. This is primarily because we opted to directly reason about sets as permutation-equivalence classes of ordered lists which have no duplicate elements, instead of using a set library like [7]. As a result, details of our set implementation have complicated our proofs. We found that in general, Ynot’s separation logic tactics were able to successfully isolate reasoning about the heap, reducing the problem of certification to a straightforward but non-trivial Coq programming task. For a more detailed discussion of engineering proofs with Ynot, see [5].

## 6 Related Work

Our approach to building certified web systems is to prove them correct by construction at development time. Alternatively, pre-existing applications can be

certified to be free of certain errors through static analysis. In [10], for instance, the authors rule out SQL injection attacks for a large fragment of PHP using an information flow analysis to ensure that tainted application inputs are never used in SQL queries without first being checked for validity. Their notion of correctness is the absence of certain classes of errors; with Ynot we can prove correctness with respect to an arbitrary logical model of application behavior, which may itself specify the absence of injection attacks. And although we have specified our logical gradebook model in Coq, specifications can be developed using special-purpose tools such as [11]. Moreover, in Ynot, reasoning is modular: interfaces themselves guarantee correctness properties; in [10], the entire program must be analyzed. Finally, automated static checking is often unsound and prone to false positives.

Jahob [21] is similar to Ynot. It allows users to write effectful Java code, which is automatically verified against a programmer specified logical model by a combination of automated theorem provers. Jahob does not use separation logic for reasoning about memory and requires a significantly larger trusted code base than Ynot. To the best of our knowledge, Jahob has never been used to certify a system like ours.

## 7 Conclusion

We learned a number of lessons in building our certified gradebook server. The first is the importance of the logical specification of application behavior. Even the most beautiful imperative algorithm will be difficult to certify if its functional model is difficult to reason about. And perhaps just as important is knowing what the specification does not capture. For instance, our networking library does not capture timeout and retry behavior and we do not model filesystem behavior, making certain applications difficult or impossible to specify without modifying the I/O library. Additionally because Hoare Logic only captures partial correctness, the divergent computation is a certified implementation of every specification.

Real-world web systems are considerably more complex than our gradebook application, and Ynot’s feasibility at larger scales is still untested. Indeed, we are only now building executable applications (rather than only datastructures) with Ynot. But given that realistic systems will invariably require imperative features, we believe our results here are a good start.

One possible future direction is to further refine the I/O library to take additional behaviors into account. Another direction is to certifying a more realistic imperative database [9]. It is likely that the results of such an effort would also be useful in certifying realistic filesystems. Finally, our server is single threaded but concurrency can be added to separation logic [4] and transactions can also be considered [16].

## References

1. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development*. Texts in Theoretical Computer Science. Springer Verlag, 2004.
2. Steve Bishop, Matthew Fairbairn, Michael Norrish, Peter Sewell, Michael Smith, and Keith Wansbrough. Engineering with logic: Hol specification and symbolic-evaluation testing for tcp implementations. In *POPL '06*, pages 55–66, New York, NY, USA, 2006. ACM.
3. Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *J. ACM*, 42(1):232–268, 1995.
4. Stephen Brookes. A semantics for concurrent separation logic. *Theor. Comput. Sci.*, 375(1-3):227–270, 2007.
5. Adam Chlipala, Gregory Malecha, Greg Morrisett, Avraham Shinnar, and Ryan Wisnesky. Effective interactive proofs for higher-order imperative programs. In *Proc. ICFP*, 2009.
6. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.
7. Jean-Christophe Filliâtre and Pierre Letouzey. Functors for proofs and programs. In *ESOP*, pages 370–384, 2004.
8. Bryan Ford. Parsing expression grammars: a recognition-based syntactic foundation. In *POPL '04*, pages 111–122, New York, NY, USA, 2004. ACM.
9. Carlos Gonzalia. Towards a formalisation of relational database theory in constructive type theory. In *RelMiCS*, pages 137–148, 2003.
10. Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, Der-Tsai Lee, and Sy-Yen Kuo. Securing web application code by static analysis and runtime protection. In *WWW '04*, pages 40–52, New York, NY, USA, 2004. ACM.
11. Daniel Jackson. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, 2002.
12. Simon L. Peyton Jones and Philip Wadler. Imperative functional programming. In *POPL '93*, pages 71–84, New York, NY, USA, 1993. ACM.
13. Wolfgang Keller. Mapping objects to tables: A pattern language. In *EuroPLOP*, 1997.
14. Andres Löb and Ralf Hinze. Open data types and open functions. In *PPDP '06*, pages 133–144, New York, NY, USA, 2006. ACM Press.
15. James Mckinna and Joel Wright. A type-correct, stack-safe, provably correct expression compiler in epigram. In *JFP*, 2006.
16. Aleksandar Nanevski, Paul Govereau, and Greg Morrisett. Towards type-theoretic semantics for transactional concurrency. In *TLDI '09*, pages 79–90, New York, NY, USA, 2008. ACM.
17. Aleksandar Nanevski, Greg Morrisett, and Lars Birkedal. Polymorphism and separation in hoare type theory. *Proc. ICFP*, 2006.
18. Aleksandar Nanevski, Greg Morrisett, Avraham Shinnar, Paul Govereau, and Lars Birkedal. Ynot: dependent types for imperative programs. In *Proc. ICFP*, 2008.
19. Gleb Naumovich and Paolina Centonze. Static analysis of role-based access control in j2ee applications. *SIGSOFT Softw. Eng. Notes*, 29(5):1–10, 2004.
20. John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Symposium on Logic in Computer Science, LICS'02*, 2002.
21. Karen Zee, Viktor Kuncak, and Martin Rinard. Full functional verification of linked data structures. In *Proc. PLDI*, 2008.



# Towards a Framework for the Verification of UML Models of Services<sup>★</sup>

Federico Banti, Rosario Pugliese, and Francesco Tiezzi

Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze  
Viale Morgagni 65, 50134 Firenze, Italia  
fbanti@gmail.com, {pugliese,tiezzi}@dsi.unifi.it

**Abstract.** We make a connection between different layers of abstraction of the engineering process of Service-Oriented Architectures (SOAs) by presenting an encoding of UML4SOA, a UML profile for modeling SOAs, in COWS, a process calculus for specifying service-oriented systems. The encoding provides a rigorous semantics for UML4SOA and paves the way for the verification of UML4SOA models by exploiting the reasoning mechanisms and analysis techniques that are available for COWS.

## 1 Introduction

Service-Oriented Architectures (SOAs) provide methods and technologies for programming and deploying software applications that can run over globally available computational network infrastructures. The most successful implementations of the SOA paradigm are probably the so called *web services*, sort of independent computational entities accessible by humans and other services through the Web. They are, in general, loosely coupled and heterogeneous, widely differing in their internal architecture and, possibly, in their implementation languages. Both stand alone web services and web service-based systems usually have requirements like, e.g., service availability, functional correctness, and protection of private data. Implementing services satisfying these requirements demands the use of rigorous software engineering methodologies that encompass all the phases of the software development process, from modelling to deployment, and exploit formal techniques for qualitative and quantitative verification of systems. The goal is to initially specify the services by exploiting a high-level modelling language and then to transform the specification towards the final deployment. This methodology should guarantee the properties of the implementation code by means of the application of formal methods to test the behavioral and quantitative properties of the specification.

As a matter of fact, UML [20] is by now a widely used modelling language for specifying software systems. It is intuitive, powerful, and extensible. Recently, a UML 2.0 profile, christened UML4SOA [17, 16], has been designed for modeling SOAs. In particular, we focus our attention on UML4SOA *activity diagrams* since they express the behavioral aspects of services, which we are mainly interested to. Inspired to WS-BPEL [19], the OASIS standard language for orchestrating web services, UML4SOA activity

---

<sup>★</sup> This work has been supported by the EU project SENSORIA, IST-2 005-016004.

diagrams integrate UML with specialized actions for exchanging messages among services, specialized structured activity nodes and activity edges for representing scopes, fault and compensation handlers. Currently, UML4SOA lacks formal semantics and methods of analysis, and must hence be regarded as an *informal* modelling language.

On the contrary, several *process calculi* for the specification of SOA systems have been recently designed (see, e.g., [15, 6, 11, 5]) that provide linguistic primitives supported by mathematical semantics, and analysis and verification techniques for qualitative and quantitative properties. To exploit previous work on process calculi, in this paper we define an encoding of UML4SOA in COWS (Calculus for the Orchestration of Web Services) [15], a recently proposed process calculus for specifying and combining services while modelling their dynamic behaviour. Indeed, in [1] we have first used UML4SOA activity diagrams to specify the behaviour of a financial service and then translated *by hand* these diagrams to COWS terms to enable a subsequent analysis phase. In that context, we experimented that the specific mechanisms and primitives of COWS are particularly suitable for encoding services specified by UML4SOA activity diagrams. In fact, this is not surprising if one consider that, like UML4SOA, COWS is inspired to WS-BPEL. The encoding we introduce in this paper formalizes those intuitions and supports a more systematic and mathematically well-founded approach to engineering of SOA systems where developers can concentrate on modelling the high-level behaviour of the system and use transformations for analysis purposes.

Besides defining a transformational semantics for UML4SOA, our encoding enables the use of the tools and methodologies developed for COWS for the analysis of UML4SOA models of services. Thus, given a service specification, one can check confidentiality properties by using the type system of [13], information flow properties using the static analysis of [3], functional properties using the logic and the model checker of [9], and quantitative properties using the stochastic extension introduced in [22].

Moreover, the encoding we propose is compositional, in the sense that the encoding of a UML4SOA activity diagram is (approximately) the COWS term resulting from the parallel composition of the encodings of its components. The encoding is thus easily expandable, applicable to large and complex real applications, and suitable for automatic implementation. In fact, we are currently developing a software tool for automatically translating UML4SOA orchestrations into COWS terms. A first prototype of the software can be downloaded from the COWS's web page (<http://rap.dsi.unifi.it/cows/>).

The rest of the paper is structured as follows. Section 2 first provides an overview of UML4SOA by means of a classical ‘travel agency’ example and then presents our proposal of a BNF-like syntax for UML4SOA. Section 3 briefly reviews COWS. Section 4 presents the COWS-based transformational semantics of UML4SOA. Finally, Section 5 touches upon comparisons with related work and directions for future developments.

## 2 An overview of UML4SOA

We start by informally presenting UML4SOA through a realistic but simplified example, illustrated in Figure 1, based on the classical ‘travel agency’ scenario.

A travel agency exposes a service that automatically books a flight and a hotel according to the requests of the user. The activity starts with a *receive* action,



a message from a client containing a request for a flight (*flightReq*) and a hotel (*hotelReq*) is received. Then, the workflow forks in two parallel branches. In the left branch, by a *send&receive* action, the flight request is sent to a flight searching service (*flightService*) and the service awaits for a response message that will be stored in the variable *flightAnswer*. As soon as this action is executed, a *compensation handler* is installed. The compensation consists of a *send* action to the flight searching service with a message asking to delete the request. The received answer is then sorted by a decision node. In the right branch, similar actions are undertaken in order to book a hotel by contacting a hotel searching service (*hotelService*). If both the answers are positive, the two branches join, the answers are forwarded to the client and the activity successfully terminates. If at least one answer is negative, an exception is raised by a *raise* action. An exception may also be raised in response to an *event* consisting of an incoming message from the client, and requiring to cancel his own request. All exceptions are caught by the *exception handler* that through the action *compensate* all triggers all the compensations installed so far in reverse order w.r.t. their completion, and notifies the client that his requests have not been fulfilled.

The syntax of UML4SOA is given in [16] by a metamodel in classical UML-style. In Table 1 we provide an alternative BNF-like syntax that is more suitable for defining an encoding by induction on the syntax of constructs. Each row of the table represents a production of the form  $\text{SYMBOL} ::= \text{ALTER}_1 \mid \dots \mid \text{ALTER}_n$ , where the non-terminal *SYMBOL* is in the top left corner of the row (highlighted by a gray background), while the alternatives  $\text{ALTER}_1, \dots, \text{ALTER}_n$  are the other elements of the row.

To simplify the encoding and its exposition we adopt some restrictions on the language. We assume that every action and scope has one incoming and one outgoing control flow edge, that a fork or decision node has one incoming edge, and that a join or merge node has one outgoing edge. These restrictions do not compromise expressivity of the language and are usually implicitly adopted by most of UML users for sake of clarity. We also omit many classical UML constructs, in particular object flows, exception handlers, expansion regions and several UML actions. The rationale for this choice is that UML4SOA offers specialized versions of most of these constructs. Regarding object flows, used for passing values among nodes, they become unnecessary since, for inter-service communications, UML4SOA relies on input and output pins, while data are shared among the elements of a scope by storing them in variables.

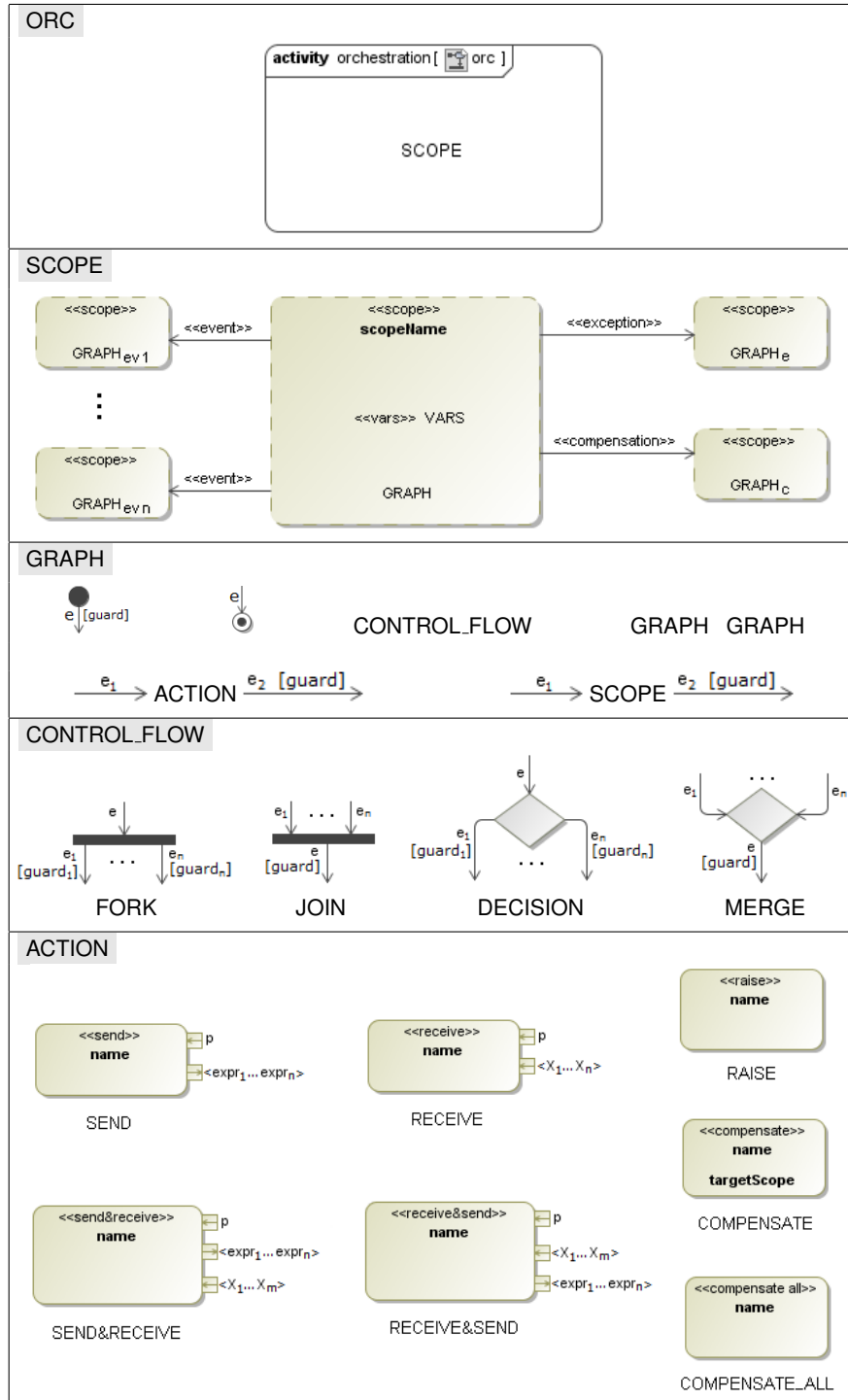
A UML4SOA *application* is a finite set of orchestrations *ORC*. We use *orc* to range over orchestration names. An *orchestration* is a UML activity enclosing one top level scope with, possibly, several nested scopes. A *scope* is a UML structured activity that permits explicitly grouping activities together with their own associated variables, references to partner services, event handlers, and a fault and a compensation handler.

A list of *variables* is generated by the following grammar:

$$\text{VARS} ::= \text{nil} \mid X, \text{VARS} \mid \ll\text{wo}\gg X, \text{VARS}$$

We use *X* to range over variables and the symbol  $\ll\text{wo}\gg$  to indicate that a variable is ‘write once’, i.e. a sort of late bound constant that can be used, e.g., to store a correlation datum (see [19, Sections 7 and 9] for further details) or a reference to a partner service. Lists of variables can be inductively built from *nil* (the empty list) by application of the





**Table 1.** UML4SOA syntax

comma operator “,”. Graphical editors for specifying UML4SOA diagrams usually permit declaring local variables as properties of a scope activity, but they are not depicted in the corresponding graphical representations. Instead, here we explicit the variables local to a scope because such information is needed for the translation in COWS. For a similar reason, we show the name of edges in the graphical representation of a graph. Notably, to obtain a compositional translation, each edge is divided in two parts: the part outgoing from the source activity and the part incoming into the target activity. In the outgoing part a guard is specified; this is a boolean expression and can be omitted when it holds true.

A *graph* GRAPH can be built by using edges to connect *initial nodes* (depicted by large black spots), *final nodes* (depicted as circles with a dot inside), control flow nodes, actions and scopes. It is worth noticing that for all incoming edges there should exist an outgoing edge with the same name, and vice-versa. Moreover, we assume that (pairs of incoming and outgoing) edges in orchestrations are pairwise distinct. These properties are guaranteed for all graphs generated by using any UML graphical editor.

Event, exception and compensation handlers are activities linked to a scope by respectively an event, a compensation and an exception activity edge. An *event handler* is a scope triggered by an event in the form of incoming message. A *compensation handler* is a scope whose execution semantically rolls back the execution of the related main scope. It is installed when execution of the related main scope completes and is executed in case of failure. An *exception handler* is an activity triggered by a raised exception whose main purpose is to trigger execution of the installed compensations.

*Default* event, exception and compensation handlers are respectively as follows: a graph composed of an initial node directly connected to a final node, a graph composed of a RAISE action preceded and followed by initial and final nodes, and a graph composed of a COMPENSATE\_ALL action preceded and followed by initial and final nodes. For readability sake, these handlers will be sometimes omitted from the representation.

It is worth noticing that, UML4SOA exception handler differs from the corresponding UML 2.0 construct. Indeed, the former can execute compensations of completed nested scopes in case of failure, while the latter can only provide an alternative way to successfully complete an activity in case an exception is raised. See Section 4 for a formal explanation of the behavior of these UML4SOA constructs.

*Control flow* nodes CONTROL\_FLOW are the standard UML ones: *fork* nodes (depicted by bars with 1 incoming edge and  $n$  outgoing edges), *join* nodes (depicted by bars with  $n$  incoming edges and 1 outgoing edge), *decision* nodes (depicted by diamonds with 1 incoming edge and  $n$  outgoing edges), and *merge* nodes (depicted by diamonds with  $n$  incoming edges and 1 outgoing edge).

Finally, UML4SOA provides seven specialized actions ACTION for exchanging data, for raising exceptions and for triggering scope compensations. SEND sends the message resulting from the evaluation of expressions  $\text{expr}_1, \dots, \text{expr}_n$  to the partner service identified by  $p$ . UML4SOA is parametric with respect to the language of the expressions, whose exact syntax is deliberately omitted; we just assume that expressions contain, at least, variables. RECEIVE permits receiving a message, stored in  $X_1, \dots, X_n$ , from the partner service identified by  $p$ . Send actions do not block the execution flow, while receive actions block it until a message is received. The other two actions for

$s ::= u \bullet u' ! \bar{e} \mid g$	(invoke, receive-guarded choice)
$\mid [e] s \mid s \mid s \mid * s$	(delimitation, parallel composition, replication)
$\mid \mathbf{kill}(k) \mid \llbracket s \rrbracket$	(kill, protection)
$g ::= \mathbf{0} \mid p \bullet o ? \bar{w}.s \mid g + g$	(empty, receive prefixing, choice)

**Table 2.** COWS syntax

message exchanging, i.e. SEND&RECEIVE and RECEIVE&SEND, are shortcuts for, respectively, a sequence of a send and a receive action from the same partner and vice-versa. RAISE causes normal execution flow to stop and triggers the associated exception handler. COMPENSATE triggers compensation of its argument scope, while COMPENSATE.ALL, only allowed inside a compensation or an exception handler, triggers compensation of all scopes (in the reverse order of their completion) nested directly within the same scope to which the handler containing the action is related.

### 3 An overview of COWS

COWS is a formalism for specifying and combining services that has been influenced by the principles underlying WS-BPEL. It provides a novel combination of constructs and features borrowed from well-known calculi such as non-binding receiving activities, asynchronous communication, polyadic synchronization, pattern matching, protection, and delimited receiving and killing activities. These features make it easier to model service instances with shared states, processes playing more than one partner role, and stateful sessions made by several correlated service interactions, inter alia.

The syntax of COWS is presented in Table 2. It is parameterized by three countable and pairwise disjoint sets: the set of (*killer*) *labels* (ranged over by  $k, k', \dots$ ), the set of *values* (ranged over by  $v, v', \dots$ ) and the set of ‘write once’ *variables* (ranged over by  $x, y, \dots$ ). The set of values is left unspecified; however, we assume that it includes the set of *names*, ranged over by  $n, m, o, p, \dots$ , mainly used to represent partners and operations. The syntax of *expressions*, ranged over by  $e$ , is deliberately omitted; we just assume that they contain, at least, values and variables, but do not include killer labels (that, hence, can *not* be exchanged in communication).

We use  $w$  to range over values and variables,  $u$  to range over names and variables, and  $e$  to range over *elements*, i.e. killer labels, names and variables. The *bar*  $\bar{\phantom{x}}$  denotes tuples (ordered sequences) of homogeneous elements, e.g.  $\bar{x}$  is a compact notation for denoting a tuple of variables as  $\langle x_1, \dots, x_n \rangle$ . We assume that variables in the same tuple are pairwise distinct. We adopt the following conventions for operators’ precedence: monadic operators bind more tightly than parallel, and prefixing more tightly than choice. We omit trailing occurrences of  $\mathbf{0}$  and write  $[e_1, \dots, e_n] s$  in place of  $[e_1] \dots [e_n] s$ . Finally, we write  $I \triangleq s$  to assign a name  $I$  to the term  $s$ .

*Invoke* and *receive* are the basic communication activities provided by COWS. Besides input parameters and sent values, both activities indicate an *endpoint*, i.e. a pair composed of a partner name  $p$  and of an operation name  $o$ , through which communication should occur. An endpoint  $p \bullet o$  can be interpreted as a specific implementation of operation  $o$  provided by the service identified by the logic name  $p$ . An invoke  $p \bullet o ! \bar{e}$  can proceed as soon as the evaluation of the expressions  $\bar{e}$  in its argument returns the corresponding values. A receive  $p \bullet o ? \bar{w}.s$  offers an invocable operation  $o$  along a given

partner name  $p$ . Execution of a receive within a *choice* permits to take a decision between alternative behaviours. Partner and operation names are dealt with as values and, as such, can be exchanged in communication (although dynamically received names cannot form the endpoints used to receive further invocations). This makes it easier to model many service interaction and reconfiguration patterns.

The *delimitation* operator is the *only* binder of the calculus:  $[e] s$  binds  $e$  in the scope  $s$ . Differently from the scope of names and variables, that of killer labels cannot be extended (indeed, killer labels are not communicable values). Delimitation can be used to generate ‘fresh’ private names (like the restriction operator of the  $\pi$ -calculus [18]) and to delimit the field of action of kill activities. Execution of a *kill* activity **kill**( $k$ ) causes termination of all parallel terms inside the enclosing  $[k]$ , which stops the killing effect. Critical activities can be protected from a forced termination by using the *protection* operator  $\{s\}$ .

Delimitation can also be used to regulate the range of application of the substitution generated by an inter-service communication. This takes place when the arguments of a receive and of a concurrent invoke along the same endpoint match and causes each variable argument of the receive to be replaced by the corresponding value argument of the invoke within the whole scope of variable’s declaration. In fact, to enable parallel terms to share the state (or part of it), receive activities in COWS do *not* bind variables (which is different from most process calculi).

Execution of *parallel* terms is interleaved, except when a kill activity or a communication can be performed. Indeed, the former must be executed *eagerly* while the latter must ensure that, if more than one matching receive is ready to process a given invoke, only one of the receives with greater priority (i.e. the receives that generate the substitution with ‘smaller’ domain, see [15] for further details) is allowed to progress. Finally, the *replication* operator  $* s$  permits to spawn in parallel as many copies of  $s$  as necessary. This, for example, is exploited to model persistent services, i.e. services which can create multiple instances to serve several requests simultaneously.

## 4 A transformational semantics for UML4SOA through COWS

Hereafter we present an encoding of UML4SOA diagrams in COWS. The encoding disambiguates the meaning of the individual diagrams. It is *compositional*, in the sense that the translation of an activity diagram is given by the (parallel) composition of the encodings of all its individual elements. We first underline the general layout, then provide specific explanations along with the presentation of the individual encodings. We refer the reader to Table 1 for the names of the encoded UML4SOA elements.

At top level, an orchestration ORC is encoded through an encoding function  $\llbracket \cdot \rrbracket$  that returns a COWS term. Function  $\llbracket \cdot \rrbracket$  is in turn defined by another encoding function  $\llbracket \cdot \rrbracket_{\text{VARS}}^{\text{orc}}$  that, given an element of a diagram, returns a COWS term and has the two additional arguments, the name *orc* of the enclosing orchestration and the names of the variables defined at the level of the encoded element. The argument *orc* is used for translating the communication activities, by specifying who is sending/receiving messages. The variable names *Vars* are necessary for delimiting the scope of the variables used by the translated element. Variables are fundamental since, as we shall show, they are

used to share received messages among the various elements of a scope and, moreover, they can also be instantiated as names of partner links.

We start by providing the encoding of the graph elements, i.e. nodes with incoming and outgoing edges, treating actions and scopes as black boxes and focusing on the encoding of passage of control among nodes. We provide then the encoding of actions, of the variables delimited within scopes and of scopes (and related handlers) themselves. We end with the translation of whole orchestrations.

*Graphs.* The encoding of a GRAPH is given simply by the parallel execution of all the COWS processes resulting from the encoding of its elements.

$$\llbracket \text{GRAPH}_1 \text{ GRAPH}_2 \rrbracket_{\text{VARS}}^{\text{orc}} = \llbracket \text{GRAPH}_1 \rrbracket_{\text{VARS}}^{\text{orc}} \mid \llbracket \text{GRAPH}_2 \rrbracket_{\text{VARS}}^{\text{orc}}$$

*Control flow nodes.* An element of a graph is encoded as a process receiving and sending signals by its incoming and outgoing edges, respectively. These edges are respectively translated as invoke and receive activities, where each edge name  $e$  is encoded by a COWS endpoint  $e$ . A guard is encoded by a COWS (boolean) expression  $\epsilon_{\text{guard}}$ . Guards are exchanged as boolean values between invoke and receive activities and the communication is allowed only if the evaluation of a guard is **true**. With the exception of initial and final nodes, the encoding of every node is a COWS process made persistent by using replication, since a node can be visited several times in the same workflow (this may occur if the activity diagram contains cycles). Practically, an initial node is translated as

$$\llbracket e \downarrow [\text{guard}] \rrbracket_{\text{VARS}}^{\text{orc}} = e! \langle \epsilon_{\text{guard}} \rangle$$

The encoding of a FORK node is a COWS service that can be instantiated by performing a receive activity corresponding to the incoming edge. After the synchronization, an invoke activity is simultaneously activated for each outgoing edge.

$$\llbracket \text{FORK} \rrbracket_{\text{VARS}}^{\text{orc}} = * e? \langle \text{true} \rangle. (e_1! \langle \epsilon_{\text{guard}_1} \rangle \mid \dots \mid e_n! \langle \epsilon_{\text{guard}_n} \rangle)$$

The encoding of a JOIN node is a service performing a sequence of receive activities, one for each incoming edge, and of an activity invoking its outgoing edge.

$$\llbracket \text{JOIN} \rrbracket_{\text{VARS}}^{\text{orc}} = * e_1? \langle \text{true} \rangle. \dots e_n? \langle \text{true} \rangle. e! \langle \epsilon_{\text{guard}} \rangle$$

The order of the receive activities does not matter, since, anyway, to complete its execution, i.e. to invoke the outgoing edge, synchronization over all incoming edges is required.

In the encoding of a DECISION node, the endpoints  $n_1, \dots, n_n$  (one for each outgoing edge) are locally delimited and used for implementing a non-deterministic guarded-choice that selects *one* endpoint among those whose guard evaluates to **true**, thus enabling the invocation of the corresponding outgoing edge.

$$\begin{aligned} \llbracket \text{DECISION} \rrbracket_{\text{VARS}}^{\text{orc}} = & * e? \langle \text{true} \rangle. [n_1, \dots, n_n] (n_1! \langle \epsilon_{\text{guard}_1} \rangle \mid \dots \mid n_n! \langle \epsilon_{\text{guard}_n} \rangle \\ & \mid n_1? \langle \text{true} \rangle. e_1! \langle \text{true} \rangle + \dots + n_n? \langle \text{true} \rangle. e_n! \langle \text{true} \rangle) \end{aligned}$$

A MERGE node is encoded as a choice guarded by all its incoming edges; all guards are followed by an invoke of its outgoing edge.

$$\llbracket \text{MERGE} \rrbracket_{\text{VARS}}^{\text{orc}} = * (e_1? \langle \text{true} \rangle. e_1! \langle \epsilon_{\text{guard}} \rangle + \dots + e_n? \langle \text{true} \rangle. e_n! \langle \epsilon_{\text{guard}} \rangle)$$

Final nodes, when reached, enable a kill activity **kill**( $k_t$ ), where the killer label  $k_t$  is delimited at scope level, that instantly terminates all the unprotected processes in the encoding of the enclosing scope (but without affecting other scopes). Simultaneously, the protected term  $t! \langle \rangle$  sends a termination signal to start the execution of (possible) subsequent activities.

$$\llbracket \text{kill} \rrbracket_{\text{VARS}}^{\text{orc}} = e? \langle \text{true} \rangle. ( \text{kill}(k_t) \mid \llbracket t! \langle \rangle \rrbracket )$$

*Action and scope nodes.* An ACTION node with an incoming and an outgoing edge is encoded as a service performing a receive on the incoming edge followed by the encoding of ACTION and, in parallel, a process waiting for a termination signal sent from the encoding of ACTION along the internal endpoint  $t$  and then performing an invoke on the outgoing edge. Of course,  $t$  is delimited to avoid undesired synchronization with other processes.

$$\llbracket \xrightarrow{e_1} \text{ACTION} \xrightarrow{e_2 [\text{guard}]} \rrbracket_{\text{VARS}}^{\text{orc}} = * e_1? \langle \text{true} \rangle. [t] ( \llbracket \text{ACTION} \rrbracket_{\text{VARS}}^{\text{orc}} \mid t? \langle \rangle. e_2! \langle \epsilon_{\text{guard}} \rangle )$$

The encoding of a SCOPE node is similar to the previous one, with two main additions. When a SCOPE terminates, the encoding of its node sends a signal  $i! \langle \rangle$  enabling the compensation related to the scope. Moreover, it sends its name to the local *Stack* process in case compensation activities are started (see the encoding of compensation handlers below for further explanations).

$$\llbracket \xrightarrow{e_1} \text{SCOPE} \xrightarrow{e_2 [\text{guard}]} \rrbracket_{\text{VARS}}^{\text{orc}} = * e_1? \langle \text{true} \rangle. [t, i] ( \llbracket \text{SCOPE} \rrbracket_{\text{VARS}}^{\text{orc}} \mid t? \langle \rangle. ( i! \langle \rangle \mid \text{stack} \cdot \text{push}! \langle \text{scopeName}(\text{SCOPE}) \rangle \mid e_2! \langle \epsilon_{\text{guard}} \rangle ) )$$

Function  $\text{scopeName}(\cdot)$ , given a scope, returns its name.

*Sending and receiving actions.* Sending and receiving actions are translated by relying on, respectively, COWS invoke and receive activities. Special care must be taken to ensure that a sent message is received *only* by the intended RECEIVE action and partner link. For this purpose, the action names are used as operation names in encoded terms. Thus, a SEND and a RECEIVE action can exchange messages only if they share the same name. Moreover, the partner name along which the communication takes place is the name *orc* of the enclosing orchestration.

Action SEND is an asynchronous call: message  $\langle \text{expr}_1, \dots, \text{expr}_n \rangle$  is sent to the partner  $p$  and the process proceeds without waiting for a reply. This is encoded in COWS by an invoke activity sending the tuple  $\langle \text{orc}, \epsilon_{\text{expr}_1}, \dots, \epsilon_{\text{expr}_n} \rangle$ , where *orc* indicates the sender of the message and will be used by the receiver to (possibly) provide a reply. The invoked partner  $p$  is rendered either as the link  $p$ , in case  $p$  is a constant, or as the COWS variable  $x_p$  in case  $p$  is a write-once variable. In parallel, a termination signal along the endpoint  $t$  is sent for allowing the computation to proceed.

$$\llbracket \text{SEND} \rrbracket_{\text{VARS}}^{\text{orc}} = \llbracket p \rrbracket_{\text{VARS}}^{\text{orc}} \cdot \text{name}! \langle \text{orc}, \epsilon_{\text{expr}_1}, \dots, \epsilon_{\text{expr}_n} \rangle \mid t! \langle \rangle$$

where  $\llbracket p \rrbracket_{\text{VARS}}^{\text{orc}}$  is  $p$  if  $\llbracket \text{wo} \rrbracket p \notin \text{VARS}$ , and  $x_p$  otherwise; similarly, each  $\epsilon_{\text{expr}_i}$  is obtained from  $\text{expr}_i$  by replacing each  $X$  in the expression such that  $\llbracket \text{wo} \rrbracket X \in \text{VARS}$  with  $x_X$ .

Unlike SEND, action RECEIVE is a blocking activity, preventing the workflow to go on until a message is received. It is encoded as a COWS receive along the endpoint  $\text{orc} \cdot \text{name}$ , with input pattern a tuple where the first element is the encoding of the link pin  $p$  and the others are either COWS variables  $x_X$  if  $\llbracket \text{wo} \rrbracket X \in \text{VARS}$  or variables  $X$  otherwise. This way, a message can be received if its correlation data match with those of the input pattern and, in this case, the other data are stored as current values of the corresponding variables.

$$\llbracket \text{RECEIVE} \rrbracket_{\text{VARS}}^{\text{orc}} = \text{orc} \cdot \text{name} ? \langle \llbracket p \rrbracket_{\text{VARS}}^{\text{orc}}, \llbracket X_1 \rrbracket_{\text{VARS}}^{\text{orc}}, \dots, \llbracket X_n \rrbracket_{\text{VARS}}^{\text{orc}} \rangle. \text{t} ! \langle \rangle$$

The encodings of actions SEND&RECEIVE and RECEIVE&SEND are basically the composition of the encodings of actions SEND and RECEIVE, and viceversa.

$$\llbracket \text{SEND\&RECEIVE} \rrbracket_{\text{VARS}}^{\text{orc}} = \llbracket \llbracket p \rrbracket_{\text{VARS}}^{\text{orc}} \cdot \text{name} ! \langle \text{orc}, \epsilon_{\text{expr}_1}, \dots, \epsilon_{\text{expr}_n} \rangle \rrbracket \\ | \text{orc} \cdot \text{name} ? \langle \llbracket p \rrbracket_{\text{VARS}}^{\text{orc}}, \llbracket X_1 \rrbracket_{\text{VARS}}^{\text{orc}}, \dots, \llbracket X_m \rrbracket_{\text{VARS}}^{\text{orc}} \rangle. \text{t} ! \langle \rangle$$

$$\llbracket \text{RECEIVE\&SEND} \rrbracket_{\text{VARS}}^{\text{orc}} = \text{orc} \cdot \text{name} ? \langle \llbracket p \rrbracket_{\text{VARS}}^{\text{orc}}, \llbracket X_1 \rrbracket_{\text{VARS}}^{\text{orc}}, \dots, \llbracket X_m \rrbracket_{\text{VARS}}^{\text{orc}} \rangle. \\ ( \llbracket \llbracket p \rrbracket_{\text{VARS}}^{\text{orc}} \cdot \text{name} ! \langle \text{orc}, \epsilon_{\text{expr}_1}, \dots, \epsilon_{\text{expr}_n} \rangle \rrbracket | \text{t} ! \langle \rangle )$$

*Actions for fault and compensation handling.* The behavior, and thus the encoding, of a RAISE is similar to that of a final node. In both cases a kill activity is enabled, in parallel with a protected termination signal invoking an exception handler. They differ for the killer label and the endpoint along which the termination signal is sent.

$$\llbracket \text{RAISE} \rrbracket_{\text{VARS}}^{\text{orc}} = \text{kill}(k_r) | \llbracket \text{r} \rrbracket ! \langle \rangle$$

In this way, a RAISE action terminates all the activities in its enclosing scope (where  $k_r$  is delimited) and triggers related the exception handler (by means of signal  $\text{r} ! \langle \rangle$ ). An exception can be propagated by an exception handler that executes another RAISE action. Notably, since default exception handlers simply execute a RAISE action and terminate, not specifying exception handlers results in the propagation of the exception to the further enclosing scope until eventually reaching the top level and thus terminating the whole orchestration.

Action COMPENSATE is encoded as an invocation of the compensation handler installed for the target scope. Action COMPENSATE\_ALL is encoded as an invocation of the local *Stack* process requiring it to execute (in reverse order w.r.t. scopes completion) all the compensation handlers installed within the enclosing scope.

$$\llbracket \text{COMPENSATE} \rrbracket_{\text{VARS}}^{\text{orc}} = c \cdot \text{scopeName} ! \langle \text{scopeName} \rangle | \text{t} ! \langle \rangle$$

$$\llbracket \text{COMPENSATE\_ALL} \rrbracket_{\text{VARS}}^{\text{orc}} = [n] ( \text{stack} \cdot \text{compAll} ! \langle n \rangle | n ? \langle \rangle. \text{t} ! \langle \rangle )$$

*Variables.* The encoding of scope variables is as follows.

$$\llbracket \text{nil} \rrbracket = \mathbf{0} \quad \llbracket X, \text{VARS} \rrbracket = \text{Var}_X | \llbracket \text{VARS} \rrbracket \quad \llbracket \llbracket \text{wo} \rrbracket X \rrbracket, \text{VARS} = \llbracket \text{VARS} \rrbracket$$

Thus, variables declared write-once (by means of  $\ll \text{wo} \gg$ ) directly corresponds to COWS variables (as we have seen, e.g., in the encoding of **SEND**). The remaining variables, i.e. variables that store values and can be rewritten several times (as usual in imperative programming languages), are encoded as internal services accessible only by the elements of the scope. Specifically, a variable  $X$  is rendered as a service  $\text{Var}_X$  providing two operations along the public partner name  $X$ : *read*, for getting the current value; *write*, for replacing the current value with a new one. When the service variable is initialized (i.e. the first time the *write* operation is used), an instance is created that is able to provide the value currently stored. When this value must be updated, the current instance is terminated and a new instance is created which stores the new value. To access the service, a user must invoke operations *read* and *write* by providing a communication endpoint for the reply and, in case of *write*, the value to be stored. Due to lack of space, the service  $\text{Var}_X$  has been omitted (we refer the interested reader to [2]).

Variables like  $X$  may (temporarily) occur in expressions used by invoke and receive activities within COWS terms obtained as result of the encoding. To get rid of these variables and finally obtain ‘pure’ COWS terms, we exploit the following encodings:

$$\begin{aligned} \langle\langle u \cdot u'! \bar{e} \rangle\rangle &= [\mathbf{m}, \mathbf{n}_1, \dots, \mathbf{n}_m] && \text{if } \bar{e} \text{ contains } X_1, \dots, X_m \\ & (X_1 \cdot \text{read}! \langle \mathbf{n}_1 \rangle \mid \dots \mid X_m \cdot \text{read}! \langle \mathbf{n}_m \rangle \\ & \mid [x_1, \dots, x_m] \mathbf{n}_1? \langle x_1 \rangle. \dots \mathbf{n}_m? \langle x_m \rangle. \mathbf{m}! \bar{e} \cdot \{X_i \mapsto x_i\}_{i \in \{1, \dots, m\}} \\ & \mid [\bar{x}] \mathbf{m}? \bar{x}. u \cdot u'! \bar{x}) \end{aligned}$$

$$\begin{aligned} \langle\langle p \cdot o? \bar{w}. s \rangle\rangle &= [x_1, \dots, x_m] && \text{if } \bar{w} \text{ contains } X_1, \dots, X_m \\ & p \cdot o? \bar{w} \cdot \{X_i \mapsto x_i\}_{i \in \{1, \dots, m\}} \cdot \\ & [\mathbf{n}_1, \dots, \mathbf{n}_m] (X_1 \cdot \text{write}! \langle x_1, \mathbf{n}_1 \rangle \mid \dots \mid X_m \cdot \text{write}! \langle x_m, \mathbf{n}_m \rangle \\ & \mid \mathbf{n}_1? \langle \rangle. \dots \mathbf{n}_m? \langle \rangle. \langle\langle s \rangle\rangle) \end{aligned}$$

where  $\{X_i \mapsto x_i\}$  denotes substitution of  $X_i$  with  $x_i$ , and endpoint  $\mathbf{m}$  returns the result of evaluating  $\bar{e}$  (of course, we are assuming that  $\mathbf{m}$ ,  $\mathbf{n}_i$  and  $x_i$  are fresh).

*Scopes.* A **SCOPE** is encoded as the parallel execution, with proper delimitations, of the processes resulting from the encoding of all its components.

$$\begin{aligned} \ll \text{SCOPE} \gg_{\text{VARS}'}^{\text{orc}} &= \\ & [\mathbf{e}, \text{stack}, \text{vars}(\text{VARS})] \\ & ([\mathbf{r}] ([k_r, k_t] (\ll \text{GRAPH} \gg_{\text{VARS}', \text{VARS}}^{\text{orc}} \mid \ll \text{Stack} \gg \\ & \mid [\mathbf{t}, k_t] \ll \text{GRAPH}_{\text{ev}1} \gg_{\text{VARS}', \text{VARS}}^{\text{orc}} \mid \dots \mid [\mathbf{t}, k_t] \ll \text{GRAPH}_{\text{ev}n} \gg_{\text{VARS}', \text{VARS}}^{\text{orc}})) \\ & \mid \mathbf{r}? \langle \rangle. \mathbf{e}! \langle \rangle) \\ & \mid \ll \text{VARS} \gg \mid \mathbf{e}? \langle \rangle. [\mathbf{t}, k_t] \ll \text{GRAPH}_e \gg_{\text{VARS}', \text{VARS}}^{\text{orc}} \\ & \mid \mathbf{i}? \langle \rangle. \ll c \cdot \text{scopeName}? \langle \text{scopeName} \rangle. [\mathbf{t}] ([k_t] \ll \text{GRAPH}_c \gg_{\text{VARS}', \text{VARS}}^{\text{orc}} \\ & \mid \mathbf{t}? \langle \rangle. \text{stack} \cdot \text{end}! \langle \text{scopeName} \rangle) \\ & \mid * [x] c \cdot \text{scopeName}? \langle x \rangle. \text{stack} \cdot \text{end}! \langle \text{scopeName} \rangle \gg) \end{aligned}$$

Function  $\text{vars}(\cdot)$ , given a list of variables  $\text{VARS}$ , returns a list of COWS variables/names, where a COWS name  $X$  corresponds to a variable  $X$  in  $\text{VARS}$ , while a COWS variable  $x_X$  corresponds to a variable  $\ll \text{wo} \gg X$  in  $\text{VARS}$ .

The (private) endpoint  $\mathbf{r}$  catches signals generated by **RAISE** actions and activate the corresponding handler, by means of the (private) endpoint  $\mathbf{e}$ . Killer labels  $k_r$  and  $k_t$



are used to delimit the field of action of kill activities generated by the translation of action *RAISE* or of final nodes, respectively, within *GRAPH*.

When a scope successfully completes, its compensation handler is installed by means of a signal along the endpoint *i*. Installed compensation handlers are protected to guarantee that they can be executed despite of any exception. Afterwards, the compensation can be activated by means of the partner name *c*. Notably, a compensation handler can be executed only once. After that, the term  $*[x] c \cdot \text{scopeName?}\langle x \rangle. \text{stack} \cdot \text{end!}\langle \text{scopeName} \rangle$  permits to ignore further compensation requests (by also taking care not to block the compensation chain).

The (protected) *Stack* service associated to a scope offers, along the partner name *stack*, three operations: *end* to catch the termination of the scope specified as argument of the operation, *push* to stack the scope name specified as argument of the operation into the associated *Stack*, and *compAll* that triggers the compensation of all scopes whose names are in *Stack*. The specification of *Stack* is as follows:

$$[q] ( \text{Lifo} \mid * [x] \text{stack} \cdot \text{push?}\langle x \rangle. q \cdot \text{push!}\langle x \rangle \\ \mid * [x] \text{stack} \cdot \text{compAll?}\langle x \rangle. [\text{loop}] ( \text{loop!}\langle \rangle \mid * \text{loop?}\langle \rangle. \text{Comp} )$$

where *loop* is used to model a while cycle executing *Comp*. The term *Comp* pops a scope name *scopeName* out of *Lifo* and invokes the corresponding compensation handler (by means of  $c \cdot \text{scopeName!}\langle \text{scopeName} \rangle$ ); in case *Lifo* is empty, the cycle terminates and a termination signal is sent along the argument *x* of the operation *compAll*.

$$\text{Comp} \triangleq [\mathbf{r}, \mathbf{e}] ( q \cdot \text{pop!}\langle \mathbf{r}, \mathbf{e} \rangle \mid [\mathbf{y}] ( \mathbf{r?}\langle \mathbf{y} \rangle. ( c \cdot \mathbf{y!}\langle \mathbf{y} \rangle \mid \text{stack} \cdot \text{end?}\langle \mathbf{y} \rangle. \text{loop!}\langle \rangle ) \\ + \mathbf{e?}\langle \rangle. \mathbf{x!}\langle \rangle ) )$$

*Lifo* is an internal queue providing ‘push’ and ‘pop’ operations. *Stack* can push and pop a scope name into/out of *Lifo* via  $q \cdot \text{push}$  and  $q \cdot \text{pop}$ , respectively. To push, *Stack* sends the value to be inserted, while to pop sends two endpoints: if the queue is not empty, the last inserted value is removed from the queue and returned along the first endpoint, otherwise a signal along the second endpoint is received. Each value in the queue is stored as a triple made available along the endpoint *h* and composed of the actual value, and two correlation values working as pointers to the previous and to the next element in the queue. The correlation value retrieved along *m* is associated with the element on top of the queue, if this is not empty, otherwise it is *empty*.

$$\text{Lifo} \triangleq [\mathbf{m}, \mathbf{h}] ( * [\mathbf{y}_v, \mathbf{y}_r, \mathbf{y}_e] ( q \cdot \text{push?}\langle \mathbf{y}_v \rangle. [\mathbf{z}] \mathbf{m?}\langle \mathbf{z} \rangle. [c] ( \mathbf{h!}\langle \mathbf{y}_v, \mathbf{z}, c \rangle \mid \mathbf{m!}\langle c \rangle ) \\ + q \cdot \text{pop?}\langle \mathbf{y}_r, \mathbf{y}_e \rangle. [\mathbf{z}] ( \mathbf{m?}\langle \mathbf{z} \rangle. [\mathbf{y}_v, \mathbf{y}_t] \mathbf{h?}\langle \mathbf{y}_v, \mathbf{y}_t, \mathbf{z} \rangle. ( \mathbf{m!}\langle \mathbf{y}_t \rangle \mid \mathbf{y}_r! \langle \mathbf{y}_v \rangle ) \\ + \mathbf{m?}\langle \text{empty} \rangle. ( \mathbf{m!}\langle \text{empty} \rangle \mid \mathbf{y}_e! \langle \rangle ) ) ) \\ \mid \mathbf{m!}\langle \text{empty} \rangle )$$

Notice that, because of the COWS’s (prioritized) semantics, whenever the queue is empty, the presence of receive  $\mathbf{m?}\langle \text{empty} \rangle$  prevents taking place of the synchronization between  $\mathbf{m!}\langle \text{empty} \rangle$  and  $\mathbf{m?}\langle \mathbf{z} \rangle$ .

*Orchestrations.* The encoding of an orchestration is that of its top-level scope.

$$\llbracket \text{ORC} \rrbracket = [k_r, c, \mathbf{t}, \mathbf{i}, \text{edges}(\text{SCOPE})] \llbracket \text{SCOPE} \rrbracket_{\text{nil}}^{\text{orc}}$$

where function  $\text{edges}(\cdot)$ , given a scope, returns the names of all the edges of the graphs contained within the scope.

## 5 Concluding remarks

We have presented an encoding of UML4SOA activity diagrams into COWS. Both languages have been defined within the European project SENSORIA [28] on developing methodologies and tools for dealing with SOAs. As far as we know, our encoding is the first (transformational) semantics of UML4SOA. It can be the cornerstone of a future framework for verification of service models specified through UML4SOA. With a similar objective, in [4] we have defined a translation from the modelling language SRML [10] into COWS.

Recently, another UML profile for SOA design, named SoaML [21], has been introduced. With respect to UML4SOA, SoaML is more focused on architectural aspects of services and relies on the standard UML 2.0 activity diagrams without further specializing them. We believe it is worth to study the feasibility of defining an encoding from SoaML into COWS, but leave it as a challenge for future work.

In this work, we focused on those constructs that are more relevant for UML4SOA, namely workflow-related constructs and the specialized UML4SOA constructs. Several works propose formal semantics for (subsets of) UML activity diagrams. Among these works the most relevant ones are those based on (extensions of) Petri Nets (see, e.g., [8, 23]). Although we regard Petri Nets as a natural choice for encoding such aspects of UML activity diagrams as workflows, other aspects turned out to be hardly representable in this formalism. For instance, in [23], the authors themselves deem the encoding of classical UML exception handling into Petri Nets as not completely satisfactory. Also, variables are not treated by the Petri Nets-based semantics of UML activity diagrams nor, at the best of our knowledge, by any other semantics.

The UMC framework [26] and the virtual machine-based approach of [7] provide operational semantics for (subsets of) UML activity diagrams by transition systems. Although these approaches are clearly less expressive, it could be interesting to compare them with the correspondent fragments of our encoding. In [25] the authors use model checking to verify a UML activity diagram of a SOA case study. In fact, the analysis is done on a *handmade* translation in UMC of the activity diagram. The authors themselves point out that an automatic translation like the one presented in this paper would be highly desirable. In [24], a stochastic semantics for a subset of UML activity diagrams is proposed. It could be interesting to compare this approach with a stochastic extension, along the line of [22], of the encoding proposed in this paper. Anyway, none of these proposals attempts to encode the UML4SOA profile and, most notably, none of them seems to be adequate for encoding its specific constructs like message exchanging actions, scopes, exceptions and compensation handlers.

We have singled COWS out of several similar process calculi for its peculiar features, specifically the termination constructs and the correlation mechanism. Kill activities are suitable for representing ordinary and exceptional process terminations, while protection permits to naturally represent exception and compensation handlers that are supposed to run after normal computations terminate. Even more crucially, the correlation mechanism (inspired by that of WS-BPEL) permits to automatically correlate messages belonging to the same interaction, preventing to mix messages from different service instances. Modelling such a feature by using session-oriented calculi designed for SOA (e.g. [5, 27, 12]) seems to be quite cumbersome. The main reason is

that UML4SOA is not session-oriented, thus the specific features of these calculi are of little help. Compared to other correlation-oriented calculi (like, e.g., [11]), COWS seems more adequate since it relies on more basic constructs and provides analysis tools and a stochastic extension.

In [16], a software tool for generating WS-BPEL code from UML4SOA models is presented. This work and the related tool had been very useful for verifying the intended meaning of UML4SOA constructs by their equivalent WS-BPEL code. However, the encoding algorithm is not capable of translating all the possible diagrams and is not compositional. Also, WS-BPEL code has not, in general, an univocal semantics and indeed the very same code generates different computations when running on different WS-BPEL implementations [14]. Thus, the proposed encoding in WS-BPEL does not provide a formal semantics to UML4SOA models.

Our long-term goal is to build a complete framework for verifying UML4SOA models. To this aim quite some work remains to be done. Currently, the encoding is implemented by a software tool (described in the full version of this paper [2]) that accepts as input a UML2 EMF XMI 2.1 file storing a UML4SOA specification. Such file can be automatically generated by the UML editor MagicDraw (<http://www.magicdraw.com>) where, to allow users to graphically specify UML4SOA activity diagrams, the UML4SOA profile (available at <http://www.mdd4soa.eu>) must be previously installed. Our tool automatically translates the XMI description into a COWS term, written in the syntax accepted by CMC [26], a model checker supporting analysis of COWS terms. As a further step in the development of a verification framework for UML4SOA models, we plan to more closely integrate the tool implementing the encoding with CMC. We also intend to investigate the challenging issue of how to tailor the (low-level) results obtained by the analysis of COWS terms to the corresponding (high-level) UML4SOA specifications, thus making the verification process as much transparent and, hence, usable as possible for developers. Other planned extensions include the encoding of a larger subset of UML 2.0 activity diagrams and its stochastic extension by relying on the stochastic variant [22] of COWS.

## References

1. F. Banti, A. Lapadula, R. Pugliese, and F. Tiezzi. Specification and analysis of SOC systems using COWS: A finance case study. In *WWV, ENTCS* 235, pp. 71–105. Elsevier, 2009.
2. F. Banti, R. Pugliese, and F. Tiezzi. Automated Verification of UML Models of Services. Tech. rep., DSI, Univ. Florence, 2009. <http://rap.dsi.unifi.it/cows/papers/uml4soa2cows.pdf>.
3. J. Bauer, F. Nielson, H.R. Nielson, and H. Pilegaard. Relational Analysis of Correlation. In *SAS, LNCS* 5079, pp. 32–46. Springer, 2008.
4. L. Bocchi, J.L. Fiadeiro, A. Lapadula, R. Pugliese, and F. Tiezzi. From Architectural to Behavioural Specification of Services. In *FESCA, ENTCS*. Elsevier, 2009. To appear.
5. M. Boreale, R. Bruni, R. De Nicola, and M. Loreti. Sessions and Pipelines for Structured Service Programming. In *FMOODS, LNCS* 5051, pp. 19–38. Springer, 2008.
6. N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and orchestration conformance for system design. In *COORDINATION, LNCS* 4038, pp. 63–81. Springer, 2006.
7. M.L. Crane and J. Dingel. Towards a UML virtual machine: implementing an interpreter for UML 2 actions and activities. In *CASCON*, pp. 96–110. ACM, 2008.

8. C. Eichner, H. Fleischhack, R. Meyer, U. Schrimpf, and C. Stehno. Compositional Semantics for UML 2.0 Sequence Diagrams Using Petri Nets. In *SDL, LNCS 3530*, pp. 133–148. Springer, 2005.
9. A. Fantechi, S. Gnesi, A. Lapadula, F. Mazzanti, R. Pugliese, and F. Tiezzi. A model checking approach for verifying COWS specifications. In *FASE, LNCS 4961*, pp. 230–245. Springer, 2008.
10. J. Fiadeiro, A. Lopes, and L. Bocchi. A Formal Approach to Service Component Architecture. In *WS-FM, LNCS 4184*, pp. 193–213. Springer, 2006.
11. C. Guidi, R. Lucchi, R. Gorrieri, N. Busi, and G. Zavattaro. SOCK: A Calculus for Service Oriented Computing. In *ICSOC, LNCS 4294*, pp. 327–338. Springer, 2006.
12. I. Lanese, F. Martins, A. Ravara, and V.T. Vasconcelos. Disciplining Orchestration and Conversation in Service-Oriented Computing. In *SEFM*, pp. 305–314. IEEE, 2007.
13. A. Lapadula, R. Pugliese, and F. Tiezzi. Regulating data exchange in service oriented applications. In *FSEN, LNCS 4767*, pp. 223–239. Springer, 2007.
14. A. Lapadula, R. Pugliese, and F. Tiezzi. A formal account of WS-BPEL. In *COORDINATION, LNCS 5052*, pp. 199–215. Springer, 2008.
15. A. Lapadula, R. Pugliese, and F. Tiezzi. A calculus for orchestration of web services. Tech. rep., DSI, Univ. Florence, 2008. <http://rap.dsi.unifi.it/cows/papers/cows-esop07-full.pdf>. An extended abstract appeared in *ESOP, LNCS 4421*, pp. 33–47, Springer.
16. P. Mayer, A. Schroeder, and N. Koch. Mdd4soa: Model-driven service orchestration. In *EDOC*, pp. 203–212. IEEE, 2008.
17. P. Mayer, A. Schroeder, and N. Koch. A model-driven approach to service orchestration. In *SCC*, pp. 533–536, 2008.
18. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, I and II. *Information and Computation*, 100(1):1–40, 41–77, 1992.
19. OASIS. Web Services Business Process Execution Language Version 2.0. Tech. rep., 2007.
20. OMG. Unified Modeling Language (UML), version 2.1.2.
21. OMG. Service oriented architecture Modeling Language (SoaML). Tech. rep., 2008.
22. D. Prandi and P. Quaglia. Stochastic COWS. In *ICSOC, LNCS 4749*, pp. 245–256. Springer, 2007.
23. H. Störrle and J.H. Hausmann. Towards a Formal Semantics of UML 2.0 Activities. In *Software Engineering, LNI 64*, pp. 117–128. GI, 2005.
24. N. Tabuchi, N. Sato, and H. Nakamura. Model-driven performance analysis of UML design models based on stochastic process algebra. In *ECMDA-FA, LNCS 3748*, pp. 41–58. Springer, 2005.
25. Maurice H. ter Beek, Stefania Gnesi, Nora Koch, and Franco Mazzanti. Formal verification of an automotive scenario in service-oriented computing. In *ICSE*, pp. 613–622. ACM, 2008.
26. M.H. ter Beek, S. Gnesi, and F. Mazzanti. CMC-UMC: A framework for the verification of abstract service-oriented properties. In *SAC*, 2009. To appear.
27. H.T. Vieira, L. Caires, and J. Costa Seco. The Conversation Calculus: A Model of Service-Oriented Computation. In *ESOP, LNCS 4960*, pp. 269–283. Springer, 2008.
28. M. Wirsing, L. Bocchi, A. Clark, J.L. Fiadeiro, S. Gilmore, M. Hözl, N. Koch, and R. Pugliese. *At your service: Service Engineering in the Information Society Technologies Program*, SENSORIA: Engineering for Service-Oriented Overlay Computers, pp. 159–182. MIT Press, 2009.

# Analyzing a Proxy Cache Server Performance Model with the Probabilistic Model Checker PRISM <sup>\*</sup>

Tamás Bérczes<sup>1</sup>, tberczes@inf.unideb.hu,  
Gábor Guta<sup>2</sup>, Gabor.Guta@risc.uni-linz.ac.at,  
Gábor Kusper<sup>3</sup>, gkusper@aries.ektf.hu,  
Wolfgang Schreiner<sup>2</sup>, Wolfgang.Schreiner@risc.uni-linz.ac.at,  
János Sztrik<sup>1</sup>, jsztrik@inf.unideb.hu

<sup>1</sup> Faculty of Informatics, University of Debrecen, Hungary, <http://www.inf.unideb.hu>

<sup>2</sup> Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz,  
Austria, <http://www.risc.uni-linz.ac.at>

<sup>3</sup> Esterházy Károly College, Eger, Hungary, <http://www.ektf.hu>

**Abstract.** We report our experience with formulating and analyzing in the probabilistic model checker PRISM a web server performance model with proxy cache server that was previously described in the literature in terms of classical queueing theory. By our work various ambiguities and deficiencies (also errors) are revealed; in particular, it is not clear how the reported paper simulates the network bandwidth, as a queue or a delay. To avoid such ambiguities we argue that nowadays performance modeling should make use of (at least be accompanied by) state machine descriptions such as those used by PRISM. On the one hand, this helps to more accurately describe the systems whose performance are to be modeled (by making hidden assumptions explicit) and give more useful information for the concrete implementation of these models (appropriate buffer sizes). On the other hand, since probabilistic model checkers such as PRISM are furthermore able to analyze such models automatically, analytical models can be validated by corresponding experiments which helps to increase the trust into the adequacy of these models and their real-world interpretation.

## 1 Introduction

The two originally distinct areas of the qualitative analysis (verification) and quantitative analysis (performance modeling) of computing systems have in the last decade started to converge by the arise of stochastic/probabilistic model checking [9]. This fact is recognized by both communities. While originally only individual authors hailed this convergence [7], today various conferences and workshops are intended to make both communities more aware of each others' achievements [4, 12]. One attempt towards this goal is to compare techniques and tools from both communities by concrete application studies. The present paper is aimed at exactly this direction.

In [1], we have shown how the probabilistic model checker PRISM [10, 8] compares favorably with a classical performance modeling environment for modeling and

---

<sup>\*</sup> Supported by the Austrian-Hungarian Scientific/Technical Cooperation Contract HU 13/2007.

analyzing retrieval queueing systems, especially with respect to the expressiveness of the models and the queries that can be performed on them. In the present paper, we are making one step forward by applying PRISM to re-assess various web server performance models with proxy cache servers that have been previously described and analyzed in the literature.

The starting point of our work is the paper [5], which presents a performance model for a system of a web server and web clients where a “proxy cache server” receives all the requests from the clients of a local network; with a certain probability the data requested by a client are already cached on the proxy server and can be returned without contacting the web server from which the data originate. The paper [5] is based on the seminal paper [11] which introduces a performance model of a web server. In [3], two of the authors of the present paper have further generalized this model by allowing the proxy cache server to receive also requests from external sources.

In this paper, we have constructed a formal model of the informal sketches in the language of PRISM [10]. This language essentially allows to construct in a modular manner a finite state transition system (thus modeling the qualitative aspects of the system) and to associate rates to the individual state transitions (thus modeling the quantitative aspects); the mathematical core of such a system is a Continuous Time Markov Chain (CTMC) which can be analyzed by the PRISM tool with respect to queries that are expressed in the language of Continuous Stochastic Logic (CSL) [9].

The remainder of this paper is structured as follows. In Section 2 we investigate the model described in [5]. First we implement it in PRISM and we try to reproduce their quantitative results. Here we only note that this article contains errors. We believe that this part is the most interesting one for the model checking community. In Section 3 we show how did we find the errors in the investigated paper by using PRISM. This section refers to our technical report [2], where more details are given. Section 4 summarizes our findings.

## 2 Performance Model of a Proxy Cache Server

The article [5] describes the model of a “proxy cache server” (PCS) to which the clients of a firm are connected such that web requests of the clients are first routed to the PCS. Referring to an illustration redrawn in Figure 1 the model can be described as follows:

Using proxy cache server, if any information or file is requested to be downloaded, first it is checked whether the document exists on the proxy cache server or not. (We denote the probability of this existence by  $p$ ). If the document can be found on the PCS then its copy is immediately transferred to the user. In the opposite case the request will be sent to the remote web server. After the requested document arrived back to the PCS then a copy of it is delivered to the user.

The solid line in Fig 1. ( $\lambda_1 = p * \lambda$ ) represents the traffic when the requested file is available on the PCS and can be delivered directly to the user. The  $\lambda_2 = (1 - p) * \lambda$  traffic depicted by dotted line, represents those requests which could not be served by the PCS, therefore these requests must be delivered from the remote web server.

If the size of the requested file is greater then the Web server’s output buffer it will start a looping process until the delivery of all requested file’s is completed. Let

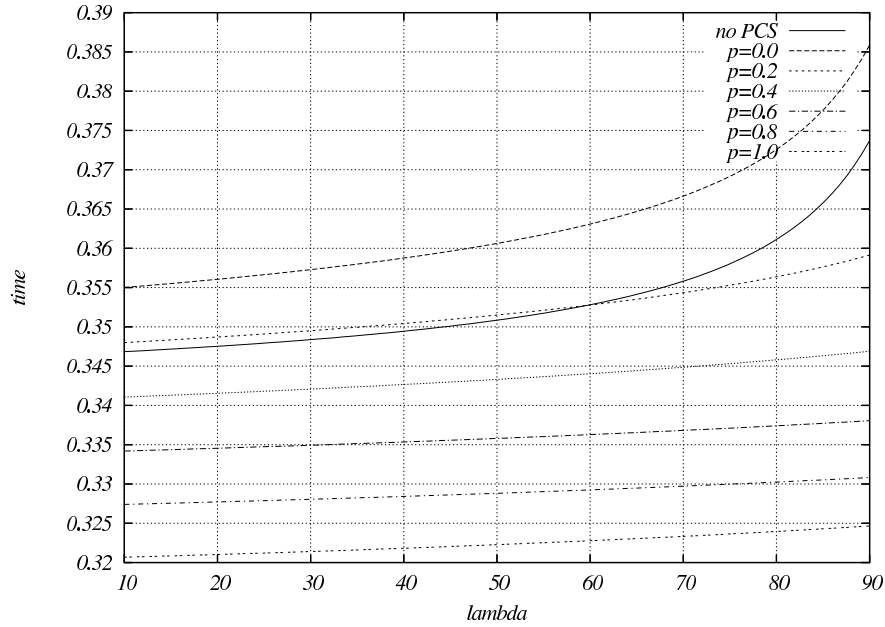


In this formula, the first term denotes the lookup time to see if the desired files are available from the PCS, the second term (with factor  $p$ ) describes the time for the content to be delivered to the requesting user, and the third term (with factor  $1 - p$ ) indicates the time required from the time the PCS initiates the fetching of the desired files to the time the PCS delivers a copy to the requesting user.

Furthermore, it is stated that without a PCS the model reduces to the special case

$$T = \frac{1}{I_s - \lambda} + \frac{1}{\frac{F}{B_s} [Y_s + \frac{B_s}{R_s}] - \lambda/q} + \frac{F}{N_s} + \frac{F}{N_c}$$

The response times for the PCS model with various arrival rates  $\lambda$  and probabilities  $p$  as well as the response time for the model without PCS, are depicted in Figure 2.



**Fig. 2.** Response Times With and Without PCS (Analytical Model)

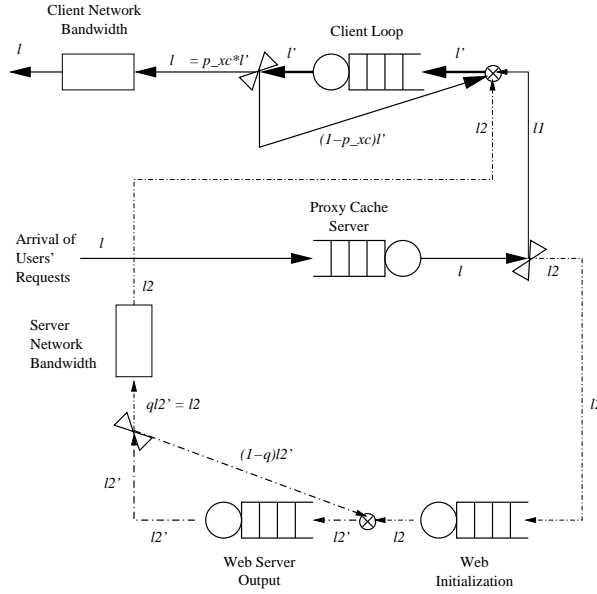
We have reconstructed this system in PRISM, and we found out that actually the two equations (for  $T$  and  $T_{xc}$ ) and also the visual model, i.e., Figure 1 are wrong. First we present the corrected equation for  $T_{xc}$ , the corrected visual model and the PRISM implementation of the corrected model. Only after this we tell how did we find those errors.



## 2.1 PRISM Implementation

First we present the corrections and the PRISM implementation of the corrected model, because we believe that this is the most interesting part of our work for the community.

Figure 3 shows the actual queueing network described in [5], but it is not the same as Figure 1, because the visual model in [5] contains errors. Figure 1 contains five queues, both the ‘server network bandwidth’ and ‘client network bandwidth’ are depicted as queues, although they are not treated as queues in the later analysis. Furthermore, it contains no queue to model the ‘client output’. More detailed description of this issue can be found in Section 3 and in our technical report [2].



**Fig. 3.** Queueing Network Model of Proxy Cache Server

The equation for the overall response time is also wrong in [5]. The correct one is:

$$T'_{xc} = \frac{1}{I_{xc} - \lambda} + p \left\{ \left( \frac{F}{B_{xc}} \right) \frac{1}{\gamma_{xc} + \frac{B_{xc}}{R_{xc}}} - \lambda / p_{xc} + \frac{F}{N_c} \right\} \\ + (1-p) \left\{ \frac{1}{I_s - \lambda_2} + \left( \frac{F}{B_s} \right) \frac{1}{\gamma_s + \frac{B_s}{R_s}} - \lambda_2 / q + \frac{F}{N_s} + \left( \frac{F}{B_{xc}} \right) \frac{1}{\gamma_{xc} + \frac{B_{xc}}{R_{xc}}} - \lambda / p_{xc} + \frac{F}{N_c} \right\}$$

where  $p_{xc} = B_{xc}/F$  is the probability that the repetition loop is terminated. The corresponding numerical results are depicted in Figure 4.

The verbal descriptions (which is however correct in [5]) gives rise to the following PRISM code which introduces by the keyword `stochastic` a continuous time Markov chain (CTMC) model [9]:

```

stochastic
...
module jobs      // generate requests at rate lambda
  [accept] true -> lambda : true ;
endmodule
module PCS      // proxy cache server
  pxwaiting: [0..IP] init 0;
  pxaccepted: bool init true;
  [accept] pxwaiting = IP -> 1 : (pxaccepted' = false);
  [accept] pxwaiting < IP -> 1 :
    (pxaccepted' = true) & (pxwaiting' = pxwaiting+1);
  [sforward] (pxwaiting > 0) & (1-p > 0) -> (1/Ixc)*(1-p) :
    (pxwaiting' = pxwaiting-1);
  [panswer] (pxwaiting > 0) & (p > 0) -> (1/Ixc)*p :
    (pxwaiting' = pxwaiting-1);
endmodule
module S_C      // client queue
  icwaiting: [0..IC] init 0;
  [panswer] icwaiting < IC -> 1 : (icwaiting' = icwaiting+1);
  [sanswer] icwaiting < IC -> 1 : (icwaiting' = icwaiting+1);
  [done] (icwaiting > 0) & (pxc > 0) -> 1/(Yxc+Bxc/Rxc)*pxc :
    (icwaiting' = icwaiting-1);
endmodule
module S_I      // server arrival queue
  waiting: [0..IA] init 0;
  [sforward] waiting < IA -> 1 : (waiting' = waiting+1);
  [forward] waiting > 0 -> (1/Is) : (waiting' = waiting-1);
endmodule
module S_R      // server output queue
  irwaiting: [0..IR] init 0;
  [forward] irwaiting < IR -> 1 : (irwaiting' = irwaiting+1);
  [sanswer] (irwaiting > 0) & (q > 0) -> 1/(Ys+Bs/Rs)*q :
    (irwaiting' = irwaiting-1);
endmodule

```

The full code is given in [2] in Appendix B.2. The model consists of one process (“module”) *jobs* generating requests and four processes *PCS*, *S<sub>C</sub>*, *S<sub>I</sub>*, and *S<sub>R</sub>*. We describe them later in this section. Each process contains declarations of its state variables (bounded integers or booleans) and state transitions of form

$$[label] \text{ guard } \rightarrow \text{ rate } : \text{ update } ;$$

A transition is enabled to execute if its *guard* condition evaluates to true; it executes with a certain (exponentially distributed) *rate* and performs an *update* on its state variables. Transitions in different processes with the same *label* execute synchronously as a single combined transition whose rate is the product of the rates of the individual transitions.

Since a product of rates rarely makes sense in a model, it is a common technique to give all but one of the individual transitions the rate 1 and let the remaining transition alone determine the combined rate (we follow this practice in all our PRISM models).

Each node models a queue with a counter, which is the number of request in the queue, i.e., we make no distinction between requests, and each node has (generally) two transitions. One (or more) for receiving requests and one (or more) for serving requests. The first one increases the counter, the second one decreases it. If two queues, say  $A$  and  $B$ , are connected, i.e., a served request from  $A$  goes to  $B$ , then the server transaction of  $A$  and the receiver transaction of  $B$  have to be synchronous, i.e., they have the same label.

The rate of the server transactions has generally this shape:  $1/t * p$ , where  $t$  is the time for processing a request and  $p$  is the probability of the branch for which the transaction corresponds. Note that if  $t$  is a time, then  $1/t$  is a rate. The rate of the receiver transactions are always 1 in this PRISM implementation, because product of rates rarely makes sense.

If a new request arrives and the queue is not full, i.e., its counter has not yet reached its upper bound, then the counter is increased and we can set an “acceptance” flag; otherwise, clear the flag (see “pxaccepted” in module  $PCS$ ). We can use this flag to approximate the acceptance ratio of the queue.

Module  $PCS$  models the proxy cache server, module  $S_C$  the client, module  $S_I$  the initialization queue of the web server, module  $S_R$  the output queue of the web server with the following behavior:

- $PCS$  returns with probability  $q$  an answer to the client (transition *answer*) and forwards with probability  $1 - q$  the request to the server (transition *sforward*). The corresponding transitions “carry” the initialization time  $I_{xc}$  of the server.
- $S_I$  buffers the incoming server request and forwards it after the initialization for further processing (transition *forward*); the transition carries the initialization time  $I_s$  of the server.
- $S_R$  generates an output buffer with rate  $1/(Y_s + \frac{B_s}{R_s})$  according to the model. However, since the request is repeated with probability  $1 - q$  (where  $q = F/B_s$ ), the final result is only produced with probability  $q$  which contributes as a factor to the rate of the corresponding transition (transition *sanswer*).
- $S_S$  models the repetition behavior of the client; a buffer of size  $B_{xc}$  is received from the PCS with rate  $1/(Y_{px} + \frac{B_{xc}}{R_{xc}})$ . However, the request for a buffer is repeated with probability  $1 - p_{xc}$  such that only with probability  $p_{xc}$  the final buffer is received and the request is completed (transition *done*).

While it would be tempting to model the repetition in  $S_C$  by generating a new request for  $PCS$ , this is actually wrong, since such a repetition request is only triggered after the PCS has already received the complete file from the web server, it is not to be treated like the incoming requests (that with probability  $1 - p$  generate requests for the web server); rather we just consider the probability  $p_{xc}$  with which the *final* block is received from the PCS in the rate of the termination transition *done*.

If we could compute  $N$ , the number of requests in the system, and  $P$ , the probability that a request is “rejected” (i.e. dropped from the system because it encounters some

full buffer), and  $\lambda$ , the arrival rate of the requests, then we could apply “Little’s Law” from queueing theory [6] to determine the average response time  $T$  for a request

$$T = \frac{N}{(1-P)\lambda}$$

Actually PRISM can be used to compute such quantitative properties of the model by using its reward system. Rewards have the form:

```
rewards "name of the reward"
  condition : numerical expression;
endrewards
```

This reward attaches to each state the value of the numerical expression where the condition is true. One can use the CSL query

```
R{"name of the reward"}=? [ S ]
```

to compute the long term average of this reward, where by operator  $R$  we introduce a reward-based property and by operator  $S$  we query the long-term average (“steady state value”) of this property.

Now we have to compute  $N$  and  $P$  ( $\lambda$  is given as a parameter of the model). For this we introduce the following rewards of the model:

```
rewards "pending"
  true : waiting + irwaiting + pxwaiting + icwaiting;
endrewards
rewards "time"
  true : (waiting + irwaiting + pxwaiting + icwaiting)/lambda;
endrewards
rewards "time0"
  true : (waiting + irwaiting + pxwaiting + icwaiting)/lambda
        + (FS/Nc) + (1-p)*(FS/Ns);
endrewards
rewards "accepted"
  pxaccepted: 1;
endrewards
```

Actually we can compute  $N$  by the reward “pending” because it assigns to every state the number of requests in the system.

It is difficult to compute  $P$ , the probability that a request is “rejected”, in PRISM. We can approximate it by using “accepted” flags, see the details in our technical report [2]. But if  $P \simeq 0$  then  $T \simeq \frac{N}{\lambda}$ . We can compute this as it is done in the reward “time”. Only one step is remaining, we have to adjust this time with the delays of the network (the network bandwidth are simulated by delays in [5]) as it is done in the reward “time0”.

Using the CSL query

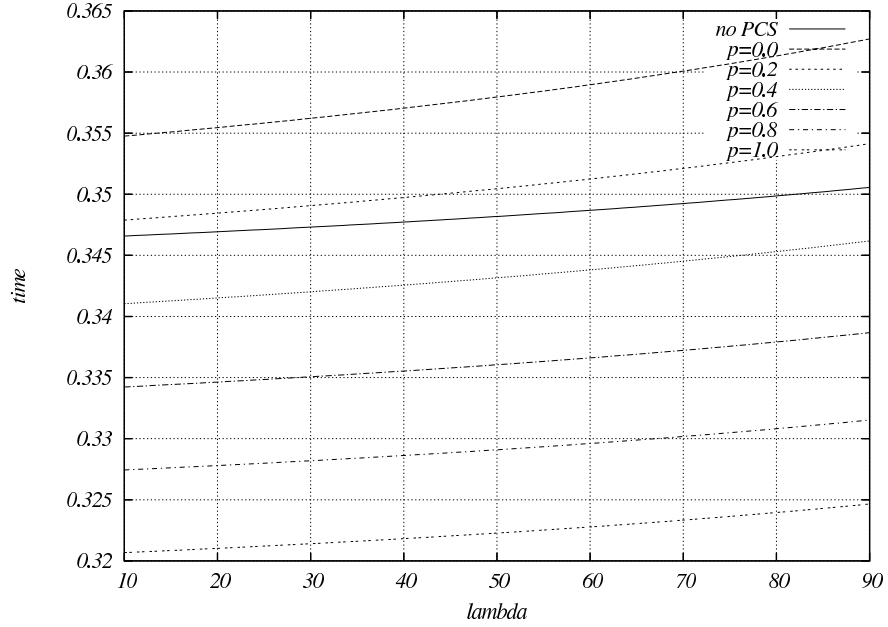
```
R{"time0"}=? [ S ]
```

we can compute the long term average of this value, which is the average response time. To be more correct, it is the average response time if the acceptance ratio for each queue is almost 1. In this paper we examine only the acceptance ratio of the PCS using the reward “accepted”.

## 2.2 Test Results

In the following, we present the results of analyzing our model in PRISM (choosing the Jacobi method for the solution of the equation systems and a relative termination epsilon of  $10^{-4}$ ; the analysis only takes a couple of seconds). As it turns out, it suffices to take the queue capacities  $IP = 5, IC = 3, IA = IR = 1$  to keep the response times essentially invariant. With this configuration the model has 192 states and 656 transitions. Note that the actual value of  $p$  and  $\lambda$  do not effect the number of states and transitions.

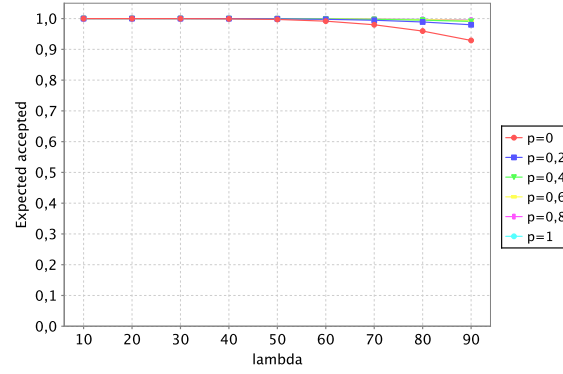
Figure 5 gives the acceptance ratio for various arrival rates  $\lambda$  and proxy hit rates  $p$ ; Figure 6 depicts the corresponding average number of requests  $N$  in the system. From this, we can estimate the total time a requests spends in the system (including the file transfer) as  $N/\lambda + \frac{F}{N_c} + (1-p)\frac{F}{N_s}$ , see Figure 7 and compare with the curve given from the equation of  $T'_{xc}$  in Figure 4. The results are virtually identical; only for arrival rates  $\lambda > 70$  and  $p = 0$ , we can see differences (because the web server gets saturated and the request rejection rate starts to get significant).



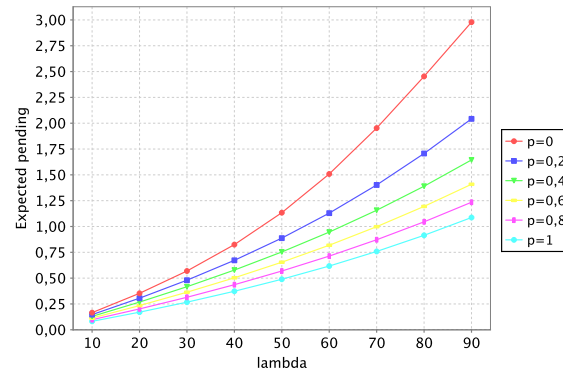
**Fig. 4.** Response Times With and Without PCS (Modified Analytical Model)

## 3 The Analytical Model Corrected

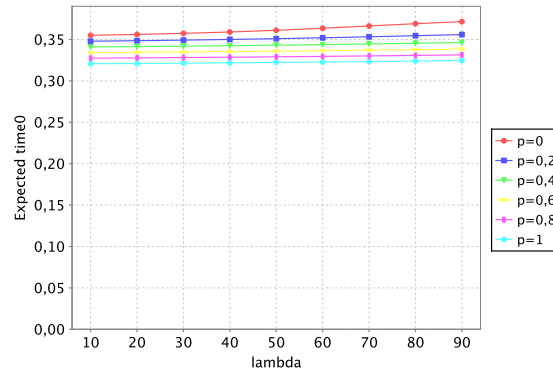
In this section we tell the “story” how could we find the errors in [5] with the help of PRISM. Variables with prime, like  $T'$ , represent the corrected equations, variables with-



**Fig. 5.** Estimated Acceptance Ratio



**Fig. 6.** Number of Pending Requests ( $N$ )



**Fig. 7.** Estimated Response Time  $N/\lambda + \frac{F}{N_c} + (1-p)\frac{F}{N_s}$

out prime, like  $T$ , represent the original equations in [5], and variables with asterisks, like  $T^*$ , represent the original equations in [11].

### 3.1 The Model without PCS

It is claimed in [5] that the equation for  $T$

$$T = \frac{1}{\frac{1}{I_s} - \lambda} + \frac{1}{\frac{F}{B_s[Y_s + \frac{B_s}{R_s}] - \lambda/q}} + \frac{F}{N_s} + \frac{F}{N_c}$$

represents the special case reported in [11], where  $T^*$  is given as

$$T^* = \frac{F}{N_c} + \frac{I_s}{1 - \lambda I_s} + \frac{F}{N_s - \lambda F} + \frac{F(B_s + R_s Y_s)}{B_s R_s - \lambda F(B_s + R_s Y_s)}$$

But this is actually *not* the case. In [5], the only term where the server bandwidth  $N_s$  plays a role is

$$\frac{F}{N_s}$$

which indicates the time for the transfer of the file over the server network. In [11], instead the term

$$\frac{F}{N_s - \lambda F}$$

is used which can be transformed to

$$\frac{1}{\frac{N_s}{F} - \lambda}$$

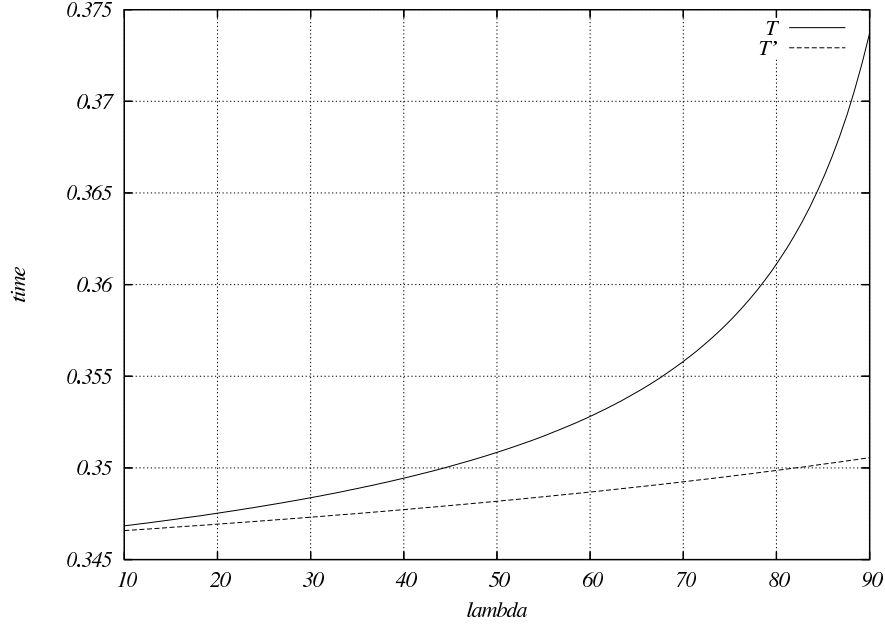
which indicates the time that a request spends in a queue with arrival rate  $\lambda$  and departure rate  $\frac{N_s}{F}$ . In other words, while [11] did not treat the client network as a queue, it nevertheless treated the server network as such. However, in [5], neither the client network nor the server network are treated as queues; they are just used to give additional time constants for file transfers.

The system without PCS can be modeled by the following PRISM implementation:

```

module jobs
  [accept] true -> lambda : true ;
endmodule
module S_I
  waiting: [0..IA] init 0;
  [accept] waiting < IA -> 1 : (waiting' = waiting+1) ;
  [forward] waiting > 0 -> (1/I_s) : (waiting' = waiting-1) ;
endmodule
module S_R
  irwaiting: [0..IR] init 0;
  [forward] irwaiting < IR -> 1 : (irwaiting' = irwaiting+1) ;
  [done] (irwaiting > 0) & (q > 0) -> 1/(Y_s+B_s/R_s)*q :
    (irwaiting' = irwaiting-1) ;
endmodule

```



**Fig. 8.** Response Time Without PCS (Modified Analytical Model)

As it turns out, the numerical results produced by the analysis in PRISM do not accurately correspond to those depicted as “No PCS” in Figure 2, in particular for  $\lambda \geq 50$ . Actually the results are better described by the equation

$$T' = \frac{1}{I_s - \lambda} + \left( \frac{F}{B_s} \right) \frac{1}{\frac{1}{Y_s + \frac{B_s}{R_s}} - \lambda/q} + \frac{F}{N_s} + \frac{F}{N_c}$$

depicted in Figure 8 where the second term (modeling the “repetition loop” in the generation of the web server output) has been modified. Indeed, a closer inspection substantiates the correctness of this formulation:  $F/B_s$  represents the number of “iterations” of the corresponding queue which has arrival rate  $\lambda/q$  and departure rate  $1/(Y_s + \frac{B_s}{R_s})$ ; this term now also equals the last term of the equation for  $T$  of [11]. (taking  $q = \frac{B_s}{F}$ ).

Actually the same problem also affects the corresponding terms in the equation  $T_{xc}$

$$T_{xc} = \frac{1}{I_{xc} - \lambda} + p \left\{ \frac{1}{\frac{F}{B_{xc}} [Y_{xc} + \frac{B_{xc}}{R_{xc}}] - \lambda_1} + \frac{F}{N_c} \right\} \\ + (1-p) \left\{ \frac{1}{I_s - \lambda_2} + \frac{1}{\frac{F}{B_s} [Y_s + \frac{B_s}{R_s}] - \lambda_2/q} + \frac{F}{N_s} + \frac{1}{\frac{F}{B_{xc}} [Y_{xc} + \frac{B_{xc}}{R_{xc}}] - \lambda_2} + \frac{F}{N_c} \right\}$$



modeling repetition loops; the correct formulation apparently is:

$$T'_{xc} = \frac{1}{I_{xc} - \lambda} + p \left\{ \left( \frac{F}{B_{xc}} \right) \frac{1}{\frac{1}{Y_{xc} + \frac{B_{xc}}{R_{xc}}} - \lambda/p_{xc}} + \frac{F}{N_c} \right\} \\ + (1-p) \left\{ \frac{1}{I_s - \lambda_2} + \left( \frac{F}{B_s} \right) \frac{1}{\frac{1}{Y_s + \frac{B_s}{R_s}} - \lambda_2/q} + \frac{F}{N_s} + \left( \frac{F}{B_{xc}} \right) \frac{1}{\frac{1}{Y_{xc} + \frac{B_{xc}}{R_{xc}}} - \lambda/p_{xc}} + \frac{F}{N_c} \right\}$$

where  $p_{xc} = B_{xc}/F$  is the probability that the repetition loop is terminated (please note also the changes in the arrival rates of the corresponding terms). The corresponding numerical results are depicted in Figure 4, compare with the original results in Figure 2. However, here the difference plays only a minor role (for  $p \geq 0.2$  only the third digit after the comma is affected).

### 3.2 The Model with PCS

Also in the model with PCS, the server network is not modeled by a queue but just by an additive constant for the transfer of the file over the network. This fact is made clear by rewriting the equation for the average response time as

$$T'_{xc} = \frac{1}{I_{xc} - \lambda} + p \left\{ \left( \frac{F}{B_{xc}} \right) \frac{1}{\frac{1}{Y_{xc} + \frac{B_{xc}}{R_{xc}}} - \lambda/p_{xc}} \right\} \\ + (1-p) \left\{ \frac{1}{I_s - \lambda_2} + \left( \frac{F}{B_s} \right) \frac{1}{\frac{1}{Y_s + \frac{B_s}{R_s}} - \lambda_2/q} + \left( \frac{F}{B_{xc}} \right) \frac{1}{\frac{1}{Y_{xc} + \frac{B_{xc}}{R_{xc}}} - \lambda/p_{xc}} \right\} \\ + \left\{ \frac{F}{N_c} + (1-p) \frac{F}{N_s} \right\}$$

Here each fraction of form  $\frac{1}{\mu - \lambda}$  indicates an occurrence of a queue with arrival rate  $\lambda$  and departure rate  $\mu$ . We can see clearly that neither the server bandwidth  $N_s$  nor the client bandwidth  $N_c$  play a role in such fractions.

Figure 1 is therefore highly misleading; neither the server network bandwidth nor the client network bandwidth are in the model actually represented by queues; thus the queues labelled as “server network bandwidth” and “client network bandwidth” should be removed (i.e. replaced by other visual elements indicating simple delays).

Furthermore, similar to the “branching” discussed in Section 2.2 of [2], the “branching” in this picture should not start after the “server network” but directly after the “web server output”, because the repetition rate of requests is not bounded by the network bandwidth in the model. To be more detailed, the server network bandwidth  $N_s$  (determining the processing rate of  $S_R$ ) only shows up in the term  $\frac{F}{N_s}$ , i.e., it is only used to contribute to the time for the transfer of the file over the server network. If indeed, as suggested by Figure 1, after the transfer of every block the server would with probability  $q$  request the transfer of another block, the maximum transfer rate of blocks ( $N_s/B_{xc} \simeq 96$ ) should also impose a limit on the number of “repetition” requests.

However, on the other side actually a queue is missing (also from the description in the text); this is the one that models the repeated requests for blocks of size  $B_{xc}$  which

are sent by the clients to the PCS (analogous to the repeated requests for blocks of size  $B_s$  sent by the client to the web server in the basic web server model); therefore the client indeed needs to be modeled by a queue (whose output is redirected with probability  $1 - p_{xc}$  to its input), but because of the looping process, not because of the client bandwidth.

Furthermore, the dotted arrow pointing to the input of the PCS queue is actually wrong; the corresponding requests do not flow to the PCS queue (where, since the queue cannot distinguish its inputs, they might generate new requests for the web server) but directly to the client queue.

Summarizing, the actual queueing network modeled in [5] contains only four nodes in contrast to the five ones shown in Figure 1 (no queue for modeling the server bandwidth) and one of these queues does not model the “client network bandwidth” but the repetition of block requests (it could be labelled in the figure as “client output” because it plays for the repetition the same role as the queue labeled “web server output”).

Figure 3 shows a revised picture that describes the model as outlined above.

## 4 Conclusions

The work described in this paper seems to justify the following conclusions:

- The informal models used in the literature for the performance analysis of computing systems are sometimes ambiguous. This may lead to misunderstandings of other researchers that build on top of prior work; e.g., [5] describes their results as to be based on the model presented in [11], but actually [11] models the server network by a delay element rather than by a queue which gives different results in the performance evaluation.
- The use of diagrams of queue networks is an insufficient substitute for a formal specification of a system model and a constant source of pitfalls. In [11], the diagram depicts a queue where the actual performance model uses a constant delay; likewise [5] depict queues for the server network but also use delays in their analysis. Furthermore, in all three papers there is an apparent confusion of the roles of the “loop-back” arrows which are shown in the diagrams in places that are misleading with respect to the role that they actually play in the analyzed models.
- The paper [5] has errors in the analytical model; these errors were only detected after trying to reproduce the results with the PRISM models. This demonstrates that performance evaluation results published in the literature cannot be blindly trusted without further validation.
- Most important, after correcting the diagrams to match the actually analyzed models, a question mark has to be put on the adequacy of the models with respect to real implementations. The papers [11, 5] model the client network bandwidth outside the “loop” for the repeated transfer of blocks from the web (respectively proxy cache) server to the client. While the informal descriptions seem to suggest that this is intended to model the underlying network protocol, i.e. presumably TCP, the “sliding windows” implementation of TCP lets the client interact with the server to control the flow of packets; this interaction is not handled in the presented

- performance models (because then the network delay must be an element of the interaction loop).
- The PRISM modeling language can be quite conveniently used to describe queueing networks by representing every network node as an automaton (“module”) with explicit (qualitative and quantitative) descriptions of the interactions between automata. This forces us to be much more precise about the system model, which may first look like a nuisance, but shows its advantage when we want to argue about the adequacy of the model.
  - The major limitation of a PRISM model is that it can be only used to model finitely bounded queues, while typical performance models use infinite queues. However, by careful experiments with increasing queue sizes one may determine appropriate bounds where the finite models do not significantly differ from the infinite models any more. Furthermore, since actual implementations typically use (for performance reasons) finite buffers anyway, such models more adequately describe the real-world situation; the work performed for the analysis may be therefore used to determine appropriate bounds for the implementations and reason about the expected losses of requests for these bounds.

## References

- [1] T. Berczes, G. Guta, G. Kusper, W. Schreiner, and J. Sztrik. Comparing the Performance Modeling Environment MOSEL and the Probabilistic Model Checker PRISM for Modeling and Analyzing Retrial Queueing Systems. Technical Report 07-17, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria, 2007.
- [2] T. Berczes, G. Guta, G. Kusper, W. Schreiner, and J. Sztrik. Analyzing Web Server Performance Models with the Probabilistic Model Checker PRISM. Technical report no. 08-17 in RISC Report Series, Johannes Kepler University Linz, Austria, 2008.
- [3] T. Berczes and J. Sztrik. Performance Modeling of Proxy Cache Servers. *Journal of Universal Computer Science*, 12(9):1139–1153, 2006.
- [4] M. Bernardo and J. Hillston, editors. *Formal Methods for Performance Evaluation*. 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007. *Lecture Notes in Computer Science*, volume 4486, 2007.
- [5] I. Bose and H. K. Cheng. Performance Models of a Firm’s Proxy Cache Server. *Decision Support Systems*, 29:47–57, 2000.
- [6] R. B. Cooper. *Introduction to Queueing Theory*. North Holland, 2nd edition, 1981.
- [7] U. Herzog. *Formal Methods for Performance Evaluation*. *Lecture Notes in Computer Science*, 2090:1–37, 2001.
- [8] A. Hinton, M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. *Lecture Notes in Computer Science, Tools and Algorithms for the Construction and Analysis of Systems*, 3920:441–444, 2006.
- [9] G. Norman, M. Z. Kwiatkowska, and D. Parker. Stochastic Model Checking. *Lecture Notes in Computer Science, Formal Methods for Performance Evaluation*, 4486:220–270, 2007.
- [10] PRISM—Probabilistic Symbolic Model Checker. [www.prismmodelchecker.org](http://www.prismmodelchecker.org). 2008.
- [11] L. P. Slothouber. A Model of Web Server Performance. *Proceedings of the 5th International World Wide Web Conference*, 1996.
- [12] K. Wolter, editor. *Formal Methods and Stochastic Models for Performance Evaluation*. Fourth European Performance Engineering Workshop, EPEW 2007. *Lecture Notes in Computer Science*, volume 4748, 2007.



# Verification of Web Content: A Case Study on Technical Documentation

Christian Schönberg, Mirjana Jakšić, Franz Weigl, and Burkhard Freitag

University of Passau, Department of Informatics and Mathematics  
94030 Passau, Germany  
{Christian.Schoenberg, Mirjana.Jaksic, Franz.Weigl,  
Burkhard.Freitag}@uni-passau.de

**Abstract.** In this paper, we present the results of a case study on a novel approach to document verification. Combining new techniques of user constraint specification and model checking, our aim is to bridge the gap between logical precision and usability, thus enabling authors and inexperienced users to employ formal verification methods. Based on a technical documentation in the form of a web document, we show that our approach is effective, efficient and has a high usability. Additionally, we argue that document verification is highly relevant for many applications, but especially for web content and hypertext documents.

## 1 Introduction

Keeping technical documentations in a consistent state – w.r.t. both structure and content – is a hard task. Many documentations today are compiled from a number of separate resources and text fragments, depending on current requirements and priorities. Online documentations complicate matters further because they usually offer more than one (linear) path through the document, rendering content consistency almost impossible to check manually.

At the same time, publishing documents online, in digital formats, is steadily gaining in importance. Most manufacturers make their technical documentations available on the web, while reusing content that is common to more than one product. This further increases the impact and relevancy of automatic document verification.

As part of the Verdikt<sup>1</sup> project [WJF09] we propose a framework that employs information extraction, temporal description logics, and model checking to verify consistency criteria on a multitude of document types. In this paper, we will describe the techniques in the context of a case study in the domain of technical documentation.

The results of the case study

- confirm the usability of our method for users not acquainted with formal methods,

---

<sup>1</sup> This work is funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) under grant number FR 1021/7-1.

- show the limited effort required to prepare a web document for model checking,
- reveal the efficiency of model checking, and
- demonstrate the precision and usefulness of error reports generated by model checking.

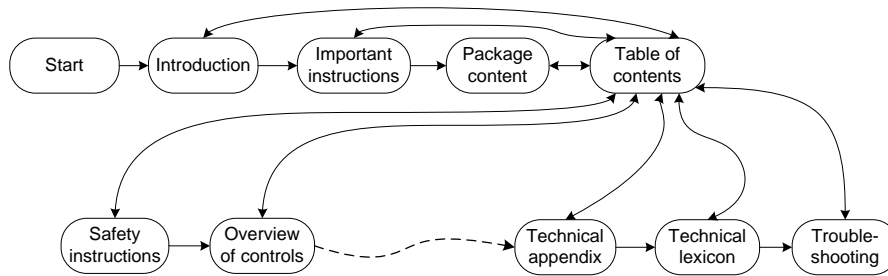
As a result, our approach helps bridging the gap between formal precision and usability. This sets it apart from existing work in the field of document verification.

For the purpose of this use case, we have adopted a version of our approach that is simplified in several areas. At the appropriate points, we will describe briefly how our framework goes beyond the techniques described here. Our approach is explained in more detail in [WJF09].

The rest of this paper is organized as follows: Section 2 describes our use case. Section 3 gives an overview of the Verdikt framework. Sections 4 through 6 contain more detailed descriptions of the main components of the framework and exemplify the overall processing based on our use case. Section 7 presents the results achieved, section 8 discusses related work, and section 9 concludes the paper.

## 2 Use Case

The case study presented in this paper concentrates on the domain of technical documentation. As a sample document we picked a manual of a digital satellite receiver published on the manufacturer’s web site and anonymized it, basically by replacing company names, product brands, images, and most distinct phrases. This lead to a document containing all the general features, characteristics, and errors of the original but being anonymous in the sense that it cannot be linked to a specific brand or seller. As a result, we obtained a document of 80 printed pages, split into 25 HTML files that is a typical representative of technical documents in terms of content, structure, and size. By convention, the content of each HTML file is called a *chapter* of the document in the sequel.



**Fig. 1.** Structure of the sample document

Fig. 1 presents a part of the basic structure of the web document in the form of a directed graph of HTML pages / chapters (vertices) and links between them (edges). The document begins with a title page (*Start*), followed by a short *Introduction*, followed by *Important instructions*, *Package content*, and a *Table of contents*. The *Table of contents* is succeeded by *Safety instructions* and *Overview of controls*. Further follow the instructions about all the settings, options, and functions of the receiver that could be used. The manual includes a *Technical appendix* – an overview of the complete configuration of the receiver, and a *Technical lexicon* – an overview of all abbreviations and technical terms with explanations. The document ends with a *Troubleshooting* chapter.

The manual contains some problems that severely limit its readability. First, inconsistent notation is used for some technical terms. For example, as a short form of the term “Conditional Access Module”, the notation “CA-Module” has been used in all chapters of the document, except for the *Technical lexicon*, where this term is referred to as “CAM”. We also found some abbreviations like “SPDIF”, “EPG”, or “FBAS”, which are not explained in the *Technical lexicon* at all. Finally, there are some interfaces (like “Ethernet”) shown in the chapter *overview of controls*, which are not described later on.

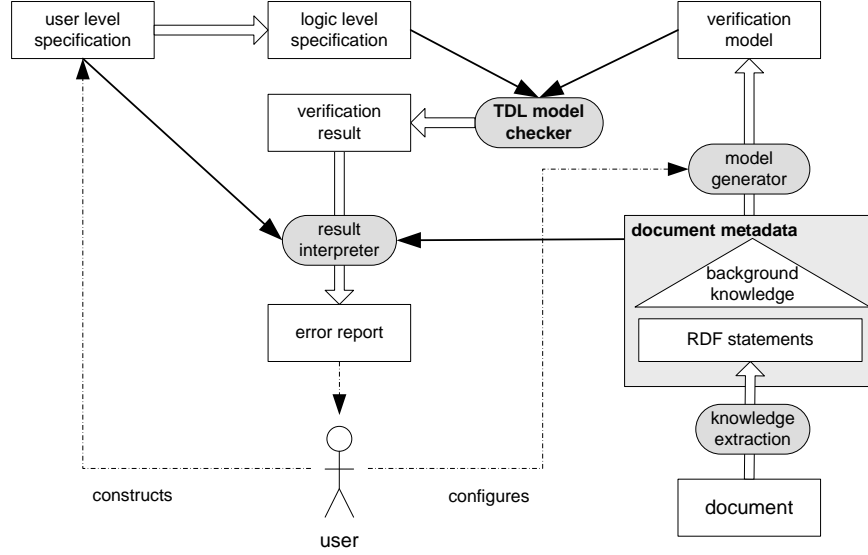
Among the objectives of this case study is to demonstrate the usability for the end user and to determine the effort required to apply our framework in a real-world scenario. Both are important factors for the general practicability of the approach. In addition, the efficiency of the system is evaluated in runtime tests.

### 3 Verdikt Framework

In the context of the Verdikt project, a general framework for document verification has been developed [WJF09], which is divided into three major components: the information extraction and model generation component, the specification and formula generation component, and the model checking and error reporting component.

The first component (information extraction) reads all relevant data from the source document, stores it for further processing, and creates a verification model suitable for model checking. The second component (specification) allows the user to specify criteria to be applied to the input document. To this end, a high level approach to the specification process based on specification patterns is used [JF08]. The third component employs temporal description logics and model checking techniques to verify the specification against the model and to track errors to their origin in the source document. Temporal description logics allow for the concise representation of consistency criteria evaluated along some or all paths through the document the reader can sensibly follow. These paths are subsequently called *reading paths*. For instance,  $Start \rightarrow Introduction \rightarrow Table\ of\ contents \rightarrow Troubleshooting$  is a reading path in the document depicted in Fig. 1.

A detailed overview of the framework is shown in Fig. 2. The user defines a set of consistency criteria to be applied to a document (top left corner). These



**Fig. 2.** Overview of the Verdikt framework

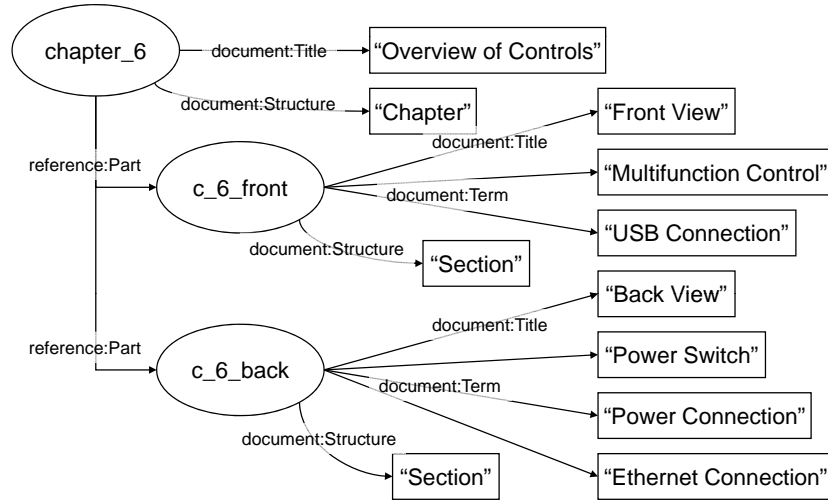
user level specifications are converted into logical formulae to be used by the model checker (top center). From a document (bottom right corner), data is extracted and stored as RDF metadata statements, which can be combined with background knowledge about the domain of discourse. The user can choose what metadata is relevant for the current application and should therefore be transferred to the verification model (top right corner), which is then passed on to the model checker. The model checking component, based on temporal description logics (TDL), calculates the verification result according to the specification and the verification model, from which an error report is distilled and presented to the user (bottom center). To make sense to the user, the error report has to take into account the original user specification and has to refer to the source document. Those source references are added to the document metadata.

## 4 Information Extraction

Extracting the necessary data from the document is performed in three stages: (a) Extract the data from the document sources (in this case HTML), (b) store the data in an RDF database, and (c) convert the data to a verification model according to the semantics of the logic level specification language  $\mathcal{ALCCTL}$  (see sections 5 and 6). The model generated in step (c) can be adjusted to meet different requirements, for example a fine-grained model describing every paragraph of the document, or a more abstract model on the level of chapters.



To facilitate creating different models without having to reprocess the original document, or to allow for ontology inferencing, the data is stored in an RDF database.



**Fig. 3.** Small extract of the RDF metadata about the sample document

(a) *HTML extraction.* First, the HTML source files are preprocessed using JTidy<sup>2</sup> to produce valid XML code. Subsequently, they are converted into RDF triples by means of an XQuery program. Using the Qexo<sup>3</sup> XQuery implementation, we developed and added further functionality to the XQuery program, including an extended context to keep track of current and previously parsed elements, list-like data structures to facilitate a file history and to avoid infinite loops, and several convenience methods to create valid RDF XML code.

Using a predefined vocabulary and CSS classes to infer structural information, a metadata description of the sample document is generated, which is represented in RDF (see Fig. 3).

Structural metadata includes information about the document’s chapter and subchapter structure, as well as the content type of text units: for example, a section is identified as an *Introduction* or as a *Technical appendix*. In our case study, evaluating stylesheet classes and rudimentary keyword analysis were sufficient to determine all important attributes. For more complex documents, we have dictionaries and thesauri at our disposal, and we have made some initial experiments based on natural language processing and machine learning [Sch08,GCW<sup>+</sup>96].

<sup>2</sup> Java HTML Tidy, © World Wide Web Consortium

<sup>3</sup> Qexo is part of GNU Kawa, © Per Bothner

Similar techniques can be applied to the extraction of content information, which is however the more difficult task. In this case study, we used background knowledge about the important terms, so we could employ elementary grammar rules to find all instances of those terms in the text.

(b) *RDF database storage.* After a comparative analysis of both the capabilities and the performance of different RDF database systems [SF09a], we decided to employ the Sesame Framework<sup>4</sup>, an open source software that supports several relational databases, including PostgreSQL and MySQL, and different query languages, including SPARQL and an extension of RQL. It both outperformed and offered better reliability than its major competitor, the Jena Framework<sup>5</sup>.

In our case study, the RDF data amounted to nearly 900 statements. This number can easily grow by one or several orders of magnitude for very large documents or interconnected web pages. In these cases, memory management becomes an issue, and storing the data in a database system is mandatory [SF09a].

(c) *Verification model generation.* We again use XQuery to generate a verification model from the RDF graph. This gives us more flexibility in customizing the verification model to the requirements of the use case than native RDF query languages would, since these do not support recursive queries required to track paths in the RDF graph [SF09a]. Instead of simply transferring the entire metadata about the document to the verification model, we exclude information irrelevant for the desired specifications. This is information that does not affect the result of verifying a given specification, because neither the information itself nor further information inferred from it are referenced in the specifications. This increases the efficiency of model checking and helps to facilitate a greater ease of use by providing a concise set of vocabulary for user level specifications.

Fig. 4 provides an example of how the verification model is generated from the document metadata w.r.t. a set of external parameters. The parameters for this example specify that the relevant structural unit is the “Chapter” (as opposed to e.g. “Section” or “Paragraph”), that any technical terms should be represented by a concept named *Term*, and that the relation between chapters and their titles and subtitles should be represented by a role named *hasTopic*. The verification model is described in detail in section 6.

## 5 Formal Specification

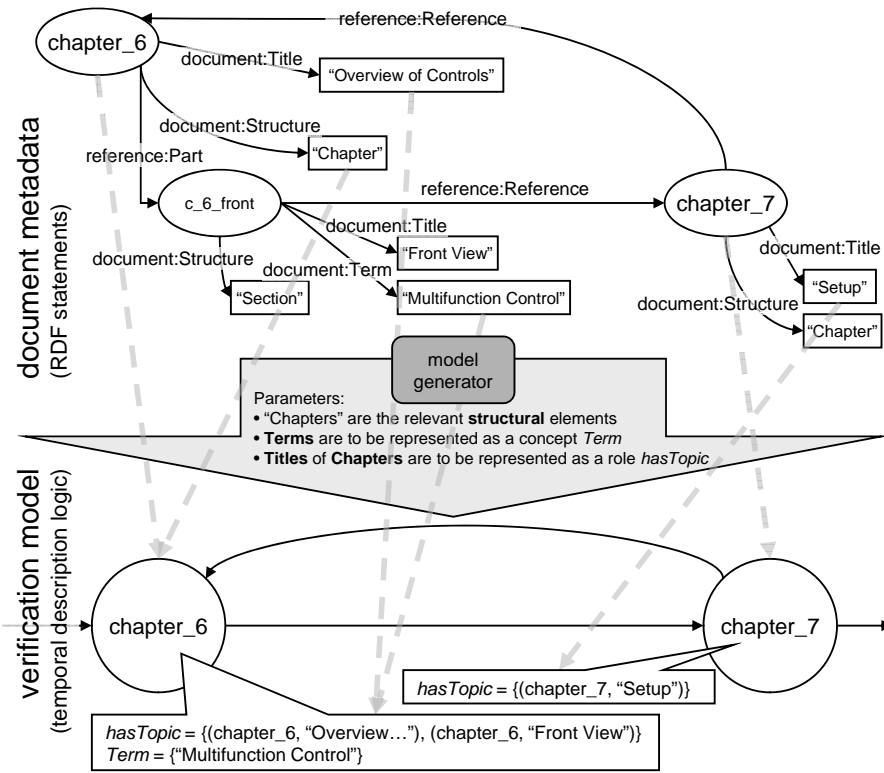
For the purpose of our case study, we have chosen the following sample consistency criteria:

1. Each abbreviation has to be defined later on.
2. Any technical term used in the document should be explained in the *technical lexicon*.

---

<sup>4</sup> © Aduna Software

<sup>5</sup> © Hewlett-Packard Development Company, LP



**Fig. 4.** Generating the verification model from the RDF metadata

3. The *safety instructions* should be listed immediately after the *table of contents*.
4. The *package content* should be listed before the *table of contents*.
5. Any interface shown in the *overview of controls* should be explained later on in the *technical appendix*.

To prove these or any other criteria automatically, they have to be expressed in some formal language. For the formal representation of consistency criteria, we use the temporal description logic  $\mathcal{ALCCTL}$ , which has been introduced in [Wei08].  $\mathcal{ALCCTL}$  is a combination of the description logic  $\mathcal{ALC}$  [BN03] and the branching time temporal logic CTL [Eme90].  $\mathcal{ALC}$  is expressive for representing structured properties of single content elements. CTL is expressive for representing loose criteria on reading paths through the document. The combination of description logics and temporal logics provides high expressiveness for content-related criteria w.r.t. reading paths.

For illustration, consider the first sample criterion: *Each abbreviation has to be defined later on*, which can be expressed in  $\mathcal{ALCCTL}$  as:

$$AbbreviatedTerm \sqsubseteq EF \text{ DefinedTerm} \quad (1)$$

Criterion 5 is represented in  $\mathcal{ALCCTL}$  as

$$OverviewOfControls \sqsubseteq \forall shownInterface. AF \exists explainedIn. TechAppendix \quad (2)$$

*AbbreviatedTerm* and *DefinedTerm* in formula (1) are concepts representing an abbreviation and a definition of a technical term, respectively.  $\sqsubseteq$  expresses that all instances of the concept to its left (in this case *AbbreviatedTerm*) are also instances of the concept to its right (in this case  $EF \text{ DefinedTerm}$ ).  $EF$  (read “some path future”) is a temporal connective representing the set of objects which on some path are eventually an instance of the concept in the scope of the  $EF$  quantification. For instance,  $EF \text{ DefinedTerm}$  is the set of terms being defined in some text unit reachable from the current text unit on some reading path.

*shownInterface* and *explainedIn* in formula (2) are description logic *roles* representing binary relations between text units and topics.

$\exists explainedIn. TechAppendix$  represents the set of objects that are explained in *some* ( $\exists$ ) *Technical appendix* of the document.

$\forall shownInterface. AF \exists explainedIn. TechAppendix$  specifies the set of objects that *only* ( $\forall$ ) show interfaces that are on all paths eventually ( $AF$ ) explained in some *Technical appendix*. For a the complete and precise definition of syntax and semantics of  $\mathcal{ALCCTL}$  we refer the reader to [Wei08]. Formulae (1) and (2) cannot be expressed by existing propositional temporal logics such as CTL or LTL [Eme90] because they include quantified expressions ( $\sqsubseteq, \forall, \exists$ ).

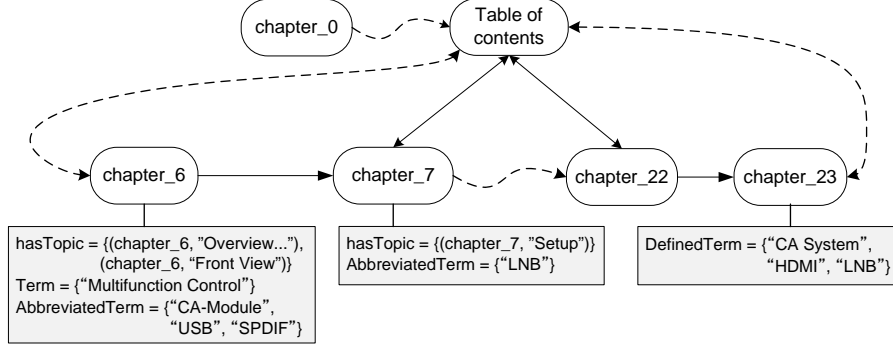
## 6 Model Checking and Error Reporting

Within our framework, specifications represented in  $\mathcal{ALCCTL}$  are verified by model checking [Wei08, WJF09].  $\mathcal{ALCCTL}$  model checking combines high precision with excellent performance [Wei08]. In the case of specification violations, counterexamples are generated that precisely pinpoint the error locations within the document.

Here, we cannot give a comprehensive introduction into model checking  $\mathcal{ALCCTL}$ . Instead, we illustrate  $\mathcal{ALCCTL}$  model checking informally on a simplified part of our use case and refer the reader to [Wei08] and [WJF09] for technical details.

The model checking problem of  $\mathcal{ALCCTL}$  is defined on top of two structures:

- a finite, non-empty set of  $\mathcal{ALCCTL}$  formulae  $F$  that represent consistency criteria to be met by a document  $d$ .
- a finite verification model  $M_d$  of document  $d$  that is derived from RDF-based metadata (Fig. 4) and represents the structure and content of document  $d$ .



**Fig. 5.** Simplified verification model of the sample document

Fig. 5 depicts a simplified part of the verification model of the presented use case. The verification model is an annotated graph  $(S, R, I)$ : Nodes  $S$  of the graph represent text units of the document (in our case: web pages), edges  $R$  represent links between text units, and annotations  $I$  represent the content of each text unit of the document.

In Fig. 5,  $S$  contains the nodes *chapter\_0* (*Start*), *Table of contents*, ... .  $R$  contains, for instance, an edge from *Table of contents* to *chapter\_7* and an edge from *chapter\_7* back to *Table of contents* (Fig. 5 center).  $I$  maps each node in  $S$  onto a set of annotations. For instance,  $I(\text{chapter\_7})$  represents the annotations for node *chapter\_7* (Fig. 5 center bottom):

$$\text{hasTopic} = \{(\text{chapter\_7}, \text{"Setup"})\} \quad (3)$$

$$\text{AbbreviatedTerm} = \{\text{"LNB"}\} \quad (4)$$

This expresses that “chapter\_7” covers the topic “Setup” and mentions the abbreviation “LNB”. The technical details of defining the interpretation  $I$  are omitted here for brevity. The respective formal definitions can be found in [Wei08].

In contrast to models of propositional temporal logics such as CTL [Eme90], the interpretation  $I$  of  $\mathcal{ALCCCTL}$  verification models allows to express relationships among objects of the modeled domain. For instance, the fact that the objects represented by *chapter\_7* and “Setup” are in a *hasTopic* relationship in node *chapter\_7* (equation (3)) cannot be represented directly by means of propositional formalism such as CTL. Being able to represent semantic interrelationships among parts and topics of a document is vital for verifying content-related properties.

The semantics of  $\mathcal{ALCCCTL}$  [Wei08] defines when an  $\mathcal{ALCCCTL}$  formula  $f$  holds at a node  $s \in S$  of a verification model  $M_d = (S, R, I)$ , in symbols  $M_d, s \models f$ .

Given a verification model  $M_d = (S, R, I)$  and an  $\mathcal{ALCCCTL}$  formula  $f$ , *model checking* is defined as determining the set

$$\text{Nodes}(M_d, f) := \{s \in S \mid M_d, s \models f\}$$

i.e. the set of nodes at which a formula  $f$  holds in a model  $M_d$  [Wei08].  $Nodes(M_d, f)$  represents the parts of the document that conform to a requirement represented by formula  $f$ .

In [Wei08] we have shown that the  $\mathcal{ALCCTL}$  model checking problem is decidable and has a polynomial runtime complexity. This is surprising because  $\mathcal{ALCCTL}$  allows quantified expressions typically leading to an exponential runtime complexity. The polynomial complexity results from the limited interaction of first order, path, and temporal quantifiers in  $\mathcal{ALCCTL}$ . These limitations do not severely affect the expressiveness of  $\mathcal{ALCCTL}$  in the given application but guarantee its efficiency also for large and complex scenarios.

We have defined a sound and complete algorithm for determining the set  $Nodes(M_d, f)$ . This algorithm runs in  $\mathcal{O}(|d|^3 \cdot |f|)$  where  $|d|$  denotes the size of the document  $d$ , and  $|f|$  denotes the size of the formula  $f$  to be verified. Documents of 5000 web pages are verified in less than 2 seconds. For comparison, the state-of-the-art CTL model checker NuSMV [CCG<sup>+</sup>02] takes 25 seconds for a document of 500 pages under similar conditions [Wei08].

For an illustration of model checking, let  $M_d$  be the verification model depicted in Fig. 5 and  $f$  be the  $\mathcal{ALCCTL}$  formula

$$AbbreviatedTerm \sqsubseteq \text{EF } DefinedTerm$$

expressing that each “abbreviated term” is on some path within the graph  $(S, R)$  eventually a “defined term” (compare section 5).

Then  $chapter\_7 \in Nodes(M_d, f)$  because for the (only) abbreviated term “LNB” in  $chapter\_7$  (Fig. 5 center bottom) there is a path to the node  $chapter\_23$  where “LNB” is a defined term (Fig. 5 rhs bottom). However,  $chapter\_6 \notin Nodes(M_d, f)$  because there are abbreviated terms “CA-Modules”, “USB”, and “SPDIF” in  $chapter\_6$  (Fig. 5 lhs bottom) that are not defined terms in any node reachable from  $chapter\_6$ .

The nodes in  $S \setminus Nodes(M_d, f)$  represent the *error locations* within document  $d$  w.r.t. formula  $f$ , i.e. the parts of the document  $d$  that do not conform to the criterion represented by formula  $f$ . By applying appropriate naming conventions, these nodes can be re-mapped easily onto the respective parts of the document. For instance, node  $chapter\_6$  represents the part of the document that is contained in  $chapter\_6.html$  (cf. Table 1, first data row).

error location	violating terms
chapter_6.html	“CA-Modules”, “USB”, “SPDIF”
chapter_10.html	“USB”, “CIM”
chapter_11.html	“CA”
...	...

**Table 1.** Error report of verifying formula  $AbbreviatedTerm \sqsubseteq \text{EF } DefinedTerm$

Based on the model checking results, an error report as sketched in Table 1 is generated. The first data row of Table 1 expresses that the terms “CA-Modules”, “USB”, and “SPDIF” used in web page *chapter\_6.html* are not satisfying the formula  $AbbreviatedTerm \sqsubseteq EF \text{ DefinedTerm}$ .

In addition to the error report, a CSS file is generated that highlights the violating terms within an error location of the document. Fig. 6 shows the pre-

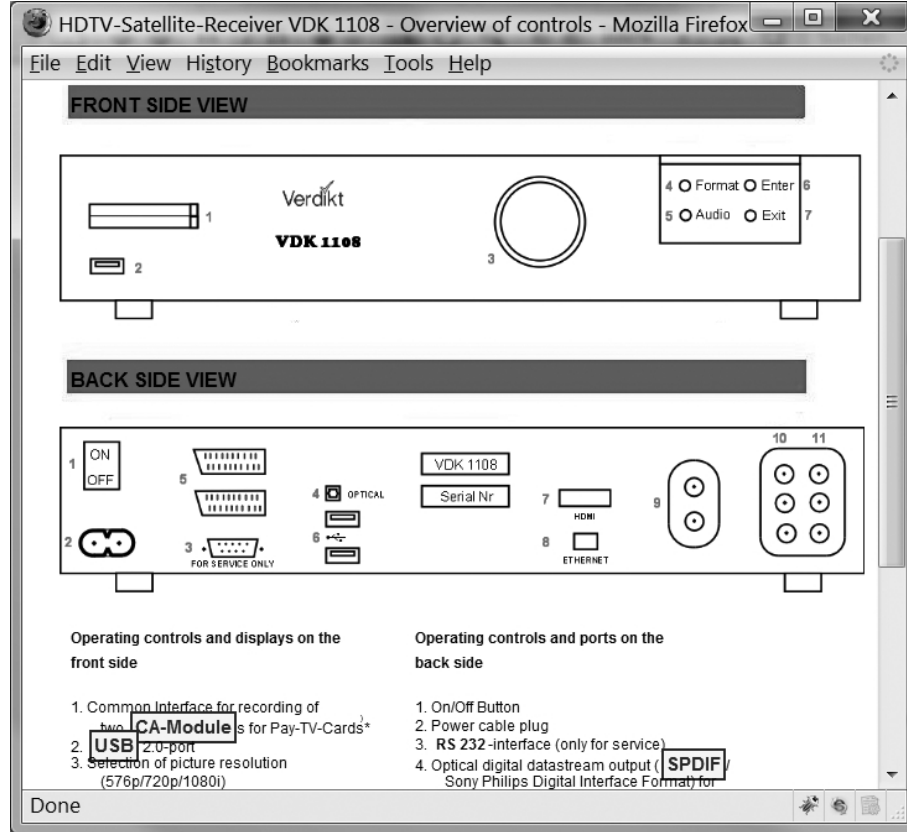


Fig. 6. Violating terms highlighted within the verified document

sented of “chapter\_6.html” with violating terms “CA-Module”, “USB”, and “SPDIF” being highlighted (bottom of Fig. 6).

## 7 Results

Table 2 summarizes the quantitative results of the case study. We checked a manual of a satellite receiver, consisting of 25 HTML files, against a set of five

# chapters of manual / HTML pages	25
# formulae	5
# violated formulae	3
# error locations	18
# violating terms	48
total runtime	9.1 s
time taken by knowledge extraction	4.4 s
model generation	4.5 s
model checking	0.1 s
report generation	0.1 s

**Table 2.** Results for verifying an online manual of a satellite receiver

criteria each of them being represented by a single  $\mathcal{ALCCCTL}$  formula (first and second row in Table 2).

18 of 25 web pages had errors (“# error locations” in Table 2). These web pages contained, in total, 48 terms that violated one of the specified properties (“# violating terms” in Table 2). This is surprising since the manual was not verified in a pre-release stage but has already been published by the manufacturer. The runtime results listed in Table 2 have been obtained on a desktop computer with Intel Pentium IV CPU at 3.2 GHz and 2 GB RAM running Windows XP and Java Version 6. The verification system has been implemented in Java.

The entire verification process took about 9 seconds (Table 2 center). The major portion of runtime was consumed by the knowledge extraction and model generation process (Fig. 2) that analyzes HTML markup, generates an RDF description of the relevant parts of the document, and finally delivers a verification model as described in sections 4 and 6. Note, however, that the verification model can be generated off-line in a preprocessing step and re-used across different verification runs.

Checking the verification model against the  $\mathcal{ALCCCTL}$ -based specification and generating an error report each took just 1% of the total runtime. Model checking and report generation scales, on average, quadratically in the size of the document [Wei08]. This ensures a quick response of the system when constructing and testing different specifications interactively.

The *application cost* for our verification system arise from

- *initial setup* of the system for a certain document format and set of target properties. As for the presented case study, it took about two hours to adjust the knowledge extraction, model generation, and error reporting components to the format of the given document. Two additional hours were required to prepare and formalize five target criteria in  $\mathcal{ALCCCTL}$ . For documents sharing the same format and target properties, the initial setup cost is independent of the size and number of documents.
- *preparing the document*. In our case study, the document has been converted from PDF to HTML format by using the HTML export function of Adobe Acrobat. The resulting HTML code has been cleaned up, anonymized, and



annotated manually, which took about eight hours. Documents in a more structured original format would require considerably less effort.

Altogether, the application cost of our verification system amounts to about 12 hours of manual effort. This initial effort amortizes quickly when a document is changed frequently or parts of it are re-used in different contexts which is typically the case for technical documentations.

The *usability* and *usefulness* of the approach were demonstrated at a large exhibition of the University of Passau targeted at the general public. Visitors who did not have any previous knowledge of verification techniques or technical documentation were able to use the system and understand its verification results after being given a brief introduction.

## 8 Related Work

Schematron [Jel02] and xlinkit [NCEF02] are powerful tools for validating the consistency of XML documents. Our approach is different from these and other XML validation techniques in the following aspects. First, our method is not limited to XML documents but can be applied to other formats, e.g. HTML, Microsoft Word, or L<sup>A</sup>T<sub>E</sub>X, to name a few (cf. [Wei08,SF09b]). Second, properties of reading paths are hard to express and inefficient to check using XPath, which is fundamental both to Schematron and xlinkit. Finally, Schematron and xlinkit are not designed to be used by authors without detailed knowledge about XML processing.

There are several approaches (e.g. [SFC98,SDM<sup>+</sup>05,FLV08]) using some propositional temporal logics (CTL, LTL), which enable the specification of complex properties along browsing paths in hypermedia structures. Even though  $\mathcal{ALCCTL}$  exceeds the expressive power of these formalisms regarding semantic relationships within the modeling domain, we nonetheless achieve a better usability by adding a user specification layer on top of the formal core. In addition, the higher expressiveness of  $\mathcal{ALCCTL}$  results in richer and more precise error reports that clearly pinpoint problems within the document.

A system for the automated verification of Web sites has been developed by the VERDI Project [ABF04]. A rule-based, formal specification language has been used to define syntactic/semantic properties of a Web site. A verification facility computes the requirements not fulfilled by the Web site, thus helping with error correction by finding incomplete or missing Web pages. However, the proposed specification language cannot express properties concerning the order of information along the reading paths through the document.

A formal consistency management component based on description logics is proposed in [ESS05] as an extension to the content management system for technical documentation Schema ST4<sup>6</sup>. Extensive tool support ensures a good usability – at least for authors experienced in technical documentation. However,

---

<sup>6</sup> <http://www.schema.de>

description logics on their own are not sufficiently expressive for representing criteria on reading paths through the document (cf. [Wei08]).

A powerful and flexible framework for checking the consistency of collections of interrelated documents has been proposed by [Sch04]. The formal basis is full first order logics interpreted over a language defined in terms of the functional programming language Haskell. While the suggested formalisms are very expressive, they are also very complex – both in terms of computation and application costs. Our approach offers a better compromise between high expressiveness and formal precision on the one hand, and efficiency, usability, and low application costs on the other.

## 9 Conclusion

We have sketched a new verification framework for web-based documents and presented the results of a case study on an online manual of a satellite receiver.

The core of the verification framework consists of flexible RDF-based meta-data representation, configurable generation of verification models, a temporal description logic as an expressive specification language, and formal verification of specifications by model checking.

The case study shows that the modularity and flexibility of the framework reduces its application cost. Temporal description logics and model checking have been demonstrated as powerful, efficient, and precise methods for the verification of web documents. Error reports generated from model checking results pinpoint error locations precisely within the document and highlight problematic terms. The system offers a high degree of usability and can – in a restricted version – be applied instantly by users without any pre-knowledge in the area of document verification. As compared to existing approaches, a better compromise between expressive power, low application costs, and usability has been found. Thus, an important step towards closing the gap between the power of formal methods and their practical applicability has been achieved. The results of this study have also raised some issues worth to be examined in more detail.

For one, while the efficiency of the model checking algorithm is already satisfactory, the efficiency of the metadata extraction process still needs to be increased. We believe that, among other things, using more native RDF database methods may help in that regard. A first prototype has shown encouraging results. To identify important terms and topics, we used a small amount of background knowledge during the extraction process. What remains to be investigated, however, is the trade-off of model quality vs. reasoning effort.

Currently, we are successively increasing the flexibility of the system without jeopardizing its usability. An intelligent specification assistant leads the user from a first vague idea of a criterion to a precise, unambiguous specification using a structured base of predefined specification patterns and application examples [JF08]. First evaluations with end users confirm that a significantly increased usability is achieved [Jak09].

## References

- [ABF04] M. Alpuente, D. Ballis, and M. Falaschi. Verdi: An automated tool for web sites verification. In *Proc. of JELIA 2004*, volume 3299 of *LNAI*, pages 726–729. Springer, 2004.
- [BN03] F. Baader and W. Nutt. Basic description logics. In *The Description Logic Handbook - Theory, Implementation and Applications*, chapter 2, pages 47–100. Cambridge University Press, 2003.
- [CCG<sup>+</sup>02] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Proc. of CAV 02*, volume 2404 of *LNCS*. Springer, 2002.
- [Eme90] E.A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science: Formal Models and Semantics*. 1990.
- [ESS05] U. Egly, B. Schiemann, and J. Schneeberger. Tech. documentation authoring based on semantic web methods. *Künstliche Intelligenz*, 2:56–59, 2005.
- [FLV08] S. Flores, S. Lucas, and A. Villanueva. Formal verification of websites. *Electronic Notes in Theoretical Computer Science*, 200:103–118, 2008.
- [GCW<sup>+</sup>96] R. Gaizauskas, H. Cunningham, Y. Wilks, P. Rodgers, and K. Humphreys. Gate: an environment to support research and development in natural language engineering. In *Proc. of the ICTAI 1996*, pages 58–66, 1996.
- [Jak09] M. Jakšić. Evaluation eines Ansatzes zur Muster-basierten Spezifikation von Konsistenzkriterien für Web-Dokumente. Technical Report MIP-0906, University of Passau, 2009.
- [Jel02] R. Jelliffe. The schematron assertion language 1.6. <http://xml.ascc.net/resource/schematron/Schematron2000.html>, 2002. last visited Feb. 2009.
- [JF08] M. Jakšić and B. Freitag. Temporal patterns for document verification. Technical Report MIP-0805, University of Passau, 2008.
- [NCEF02] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: a consistency checking and smart link generation service. *ACM Transactions on Internet Technology (TOIT)*, 2(2):151–185, 2002.
- [Sch04] J. Scheffczyk. *Consistent Document Engineering*. Dissertation, Universität der Bundeswehr München, 2004.
- [Sch08] L. Scharinger. ExtraValid - Evaluation von Methoden und Werkzeugen der Informations-Extraktion zur automatischen Annotation von Dokumenten. Diplomarbeit, Universität Passau, 2008.
- [SDM<sup>+</sup>05] E. Di Sciascio, F. M. Donini, M. Mongiello, R. Totaro, and D. Castelluccia. Design verification of web applications using symbolic model checking. In *Proc. of ICWE 2005*, volume 3579 of *LNCS*, pages 69–74. Springer, 2005.
- [SF09a] C. Schönberg and B. Freitag. Evaluating RDF querying frameworks for document metadata. Technical Report MIP-0903, Univ. of Passau, 2009.
- [SF09b] C. Schönberg and B. Freitag. Extracting and storing document metadata. Technical report, University of Passau, 2009. to appear.
- [SFC98] P. D. Stotts, R. Furuta, and C. R. Cabarrus. Hyperdocuments as automata: Verification of trace-based browsing properties by model checking. *Information Systems*, 16(1):1–30, 1998.
- [Wei08] F. Weigl. *Document Verification with Temporal Description Logics*. PhD thesis, University of Passau, 2008.
- [WJF09] F. Weigl, M. Jakšić, and B. Freitag. Towards the automated verification of semi-structured documents. *Journal of Data & Knowledge Engineering*, 68:292–317, 2009.



# A Query Language for OWL based on Logic Programming<sup>\*</sup>

Jesús M. Almendros-Jiménez

Dpto. Lenguajes y Computación.  
Universidad de Almería. jalmen@ual.es

**Abstract.** In this paper we investigate how to use logic programming (in particular, Prolog) as query language against OWL resources. Our query language will be able to retrieve data and meta-data about a given OWL based ontology. With this aim, firstly, we study how to define a query language based on a fragment of Description Logic, then we show how to encode the defined query language into Prolog by means of logic rules and finally, we identify Prolog goals which correspond to queries.

## 1 Introduction

*OWL (Web Ontology Language)* is an ontology language based on the so-called *Description Logic (DL)* [W3C04a,Bor96]. Description logic is a subset of *First Order Logic (FOL)*. OWL is a language with different fragments named *OWL Full*, *OWL DL*, *OWL Lite* and *OWL Flight*, among others. Such fragments are restricted in expressive power in order to retain reasoning capabilities and decidability. OWL Full contains all the constructors of the OWL and allows the arbitrary combination of those constructors. OWL Full semantics is an extension of *Resource Description Framework (RDF)* semantics [W3C04b], however it yields to an undecidable reasoning language [HPSvH03]. Therefore reasoning in OWL Full can be incomplete. OWL DL and OWL Lite are subsets of OWL Full in which some restrictions are considered. Therefore the RDF triple-based encoding of OWL DL and OWL Lite impose some restrictions about the RDF graphs. OWL Flight [dBLPF05] is based also in OWL, but the semantics is grounded in logic programming rather than description logic.

In this paper we investigate how to use logic programming (in particular, Prolog) as query language against OWL resources. Our query language will be able to retrieve data and meta-data about a given OWL based ontology. With this aim, firstly, we study how to define a query language based on a fragment of Description Logic, then we show how to encode the defined query language into Prolog by means of logic rules and finally, we identify Prolog goals which correspond to queries. Basically, our work goes towards the use of logic programming as query language for the *Semantic Web*. It follows our

---

<sup>\*</sup> This work has been partially supported by Spanish MICINN under grant TIN2008-06622-C03-03

research line about the use of logic programming for the handling of Web data [ABE08,ABE06,ABE09,Alm08,Alm09b,Alm09a].

In the Semantic Web, RDF(S)/OWL resources contain *data* and *meta-data* about a certain domain of interest. Such resources should be exploited by considering reasoning and inference mechanisms. Logic programming is a suitable framework for reasoning and inference. Logic programming can be used as query language for databases like in the case of the *Datalog* language. However, the structure RDF(S)/OWL resources and the underlined reasoning mechanism of RDF(S) and OWL, needs a particular treatment by means of logic programming. A query language based on logic programming to be suitable for OWL resources should include complex reasoning based on the Description Logic, which is the basis of the OWL language. In addition, RDF(S) and OWL are an special case of database in which meta-data (i.e. the database schema) and data (i.e. the database instance) are mixed, and therefore a suitable query language should retrieve not only data but meta-data. Meta-data retrieval is vital for accessing to Web data.

Our framework follows the research line about handling of OWL by means of logic programming. In this area, some authors [GHVD03,Vol04] have studied the intersection of OWL and logic programming, in other words, which fragment of OWL can be expressed in logic programming. They have defined the so-called *Description Logic Programming*, which is the intersection of logic programming and DL. Such intersection can be detected by encoding OWL into logic programming. With this aim, firstly, the corresponding fragment of OWL is represented by means of DL, after such fragment of the DL can be encoded into a fragment of FOL; finally, the fragment of FOL can be encoded into logic programming. Several fragments of OWL/DL can be encoded into logic programming, in particular, Volz [Vol04] has encoded OWL subsets into *Datalog*, *Datalog(=)*, *Datalog(=,IC)* and *Prolog(=,IC)*; where “=” means “with equality”, and “IC” means “with Integrity constraints”. Some recent proposals have encoded description logic fragments into *disjunctive Datalog* [HMS07], and into *Datalog(IC,≠,not)* (for OWL-Flight) [dBLPF05], where “not” means “with negation”.

Description logic is a formalism for expressing relationships between *concept* and *role names*, and between *concepts*, *roles* and *individuals*. Formulas of description logic can be used for representing knowledge, that is, concept descriptions, about a domain of interest. Typically, description logic is used for representing a **TBox** (*terminological box*) and the **ABox** (*assertional box*). The **TBox** describes concept (and role) *hierarchies* (i.e., relations between concepts and roles) while the **ABox** contains relations between individuals, concepts and roles. Therefore we can see the **TBox** as the meta-data description, and the **ABox** as the description about data. The most typical *decision problems*, with regard to a given ontology, include *instance checking*, that is, whether a particular individual is a member of a given concept, and *relation checking*, that is, whether two individuals hold a given role, *subsumption*, that is, whether a con-

cept is a subset of another concept, and *concept consistency*, that is, consistency through a chain of concept relationships.

However, we believe that an interesting extension of such research line would be to consider a *query language* in which *answers* will give us *the names of concepts, roles and individuals* for which a given *description logic formula is satisfied*. Such procedure of search of answers can be seen as a decision procedure in which a *description logic formula contains free variables*. Adding free variables to description logic formulas, variables can represent individuals, or even concepts and roles, and as query language they represent results of queries. In other words, a description logic-based query language could be used for obtaining either the concepts in which a given individual is included, or the roles an individual plays. Therefore our proposed query language is able to retrieve not only *data* but also *meta-data* about a given ontology. In addition, the concepts involved in such queries can be “*complex concepts*”, in the sense they can be defined by means of complex description logic formulas. Our proposed query language is also able to answer about the typical decision problems: instance checking, relation checking, subsumption and consistency problems, using Prolog as reasoning language. The reasoning mechanism is however *limited by Prolog semantics*. In addition, using Prolog command line, when the goal succeeds one obtain class, property and individual names as answers, retrieving once at a time. Our proposed query language will be able to decide about instance checking for complex concepts like  $\exists P.C(A)$ . In addition, we are able to handle the concept  $C$  and the role  $P$  in such query as the result, in the sense that, we will obtain values for  $C$  or  $P$  when  $A$  is fixed, admitting to fix  $C$  or  $P$ . Subsumption is also handled in our query language including queries about *complex relations* between concepts like  $\exists P.C \sqsubseteq D$ , in which any element (i.e. concepts  $D$ ,  $C$  or role  $P$ ) can be taken as query result.

Most of DL reasoners (for instance, *Racer* [HM01], FaCT++ [TH06], Pellet [SPG<sup>+</sup>07]) can handle the typical decision problems, and they are based on tableaux based decision procedures. Similar to our approach the tool KAON2 [HMS07] is based on logic programming. It encodes an OWL-DL ontology into a *disjunctive datalog program*. The encoding can be combined with additional logic rules, whenever they are *DL-safe* [HMS07]. KAON2 can express queries by using the SPARQL query language [dLC06]. It allows to query about the typical decision problems, but KAON2 does not allow to query about meta-data in the sense of our proposal. It is due to variables at predicate positions (representing concepts and roles) are not supported. We have studied a query language based on a fragment of DL. Although the fragment of DL of our proposal is not as powerful as the supported in KAON2, however our query language is more powerful than the KAON2 query language, including queries with concepts and roles as query results. The Protégé tool [GMF<sup>+</sup>03] is an example of tool in which reasoning includes “to find all classes that a given individual belongs to”. However, our approach extends such kind of reasoning to a more general framework.

Finally, we believe that one of the advantages of our proposal is that our query language could be used for querying OWL resources by means of a relational

database management system (RDBMS). Some recent works are concerned with the handling of ontologies in a RDBMS (see [JCJ<sup>+</sup>07,MWL<sup>+</sup>08,AKK07]). As a consequence of this research area, some tools (for instance, Jena [Jen08], Sesame [BKvH02], OWLIM [KOM05] and Oracle Semantic Technologies [WED<sup>+</sup>08], DBOWL [dMRA08], among others) are able to handle OWL in a RDBMS. Our approach is more similar to the approach of the Minerva tool [ZML<sup>+</sup>06], in the sense that they follow a logic based encoding and inference process in order to obtain all the OWL triples to be stored in a RDBMS. However, the query language of Minerva is not still able to fully handle description formulas. The Minerva tool is not able to handle complex concepts as query results. It is due to DL formulas involving complex concepts are represented in the Minerva tool by means of relational tables and therefore it cannot handle complex concepts as values.

The structure of the paper is as follows. Section 2 will present the fragment of DL of our proposal. Section 3 will define the query language defined from the fragment. Section 4 will describe the encoding of the query language in Prolog. Section 5 will introduce the RDBMS implementation of our proposal. Finally, Section 6 will conclude and present future work.

## 2 Web Ontology Language

In this section we will show what kind of ontologies will be allowed in our framework. Such kind of ontologies can be mapped into a fragment of description logic. We restrict our work to the case of a fragment of DL which can be encoded into Prolog. The study of other fragments of OWL (in the line of [Vol04,HMS07,dBLPF05]) which can be also encoded into logic programming is considered as future work. An ontology  $\mathcal{O}$  in our framework contains a **TBox**  $\mathcal{T}$  including a sequence of definitions  $\mathcal{T}_1, \dots, \mathcal{T}_n$  of the form:

$\mathcal{R} ::=$	
$C \sqsubseteq D$	( <code>rdfs:subClassof</code> )
$E \equiv F$	( <code>owl:equivalentClass</code> )
$P \sqsubseteq Q$	( <code>rdfs:subPropertyOf</code> )
$P \equiv Q$	( <code>owl:equivalentProperty</code> )
$P \equiv Q^-$	( <code>owl:inverseOf</code> )
$P \equiv P^-$	( <code>owl:SymmetricProperty</code> )
$P^+ \sqsubseteq P$	( <code>owl:TransitiveProperty</code> )
$\top \sqsubseteq \forall P^-.D$	( <code>rdfs:domain</code> )
$\top \sqsubseteq \forall P.D$	( <code>rdfs:range</code> )

where  $E, F$  are *class (i.e. concept) descriptions* of equivalence type (denoted by  $E, F \in \mathcal{E}$ ) of the form:

$$\overline{E} ::= A \mid E_1 \sqcap E_2 \mid \exists P.\{O\}$$

In addition,  $C$  is a *class (i.e. concept) description of left-hand side type* (denoted by  $C \in \mathcal{L}$ ), of the form:

$$\overline{C} ::= A \mid C_1 \sqcap C_2 \mid \exists P.\{O\} \mid C_1 \sqcup C_2 \mid \exists P.C$$

and  $D$  is a *class (i.e. concept) description of right-hand side type* (denoted by  $D \in \mathcal{R}$ ), of the form:



**Fig. 1.** Encoding of DL into FOL

$fol(C \sqsubseteq D)$	$= \forall x. fol_x(C) \rightarrow fol_x(D)$	$fol_x(A)$	$= A(x)$
$fol(E \equiv F)$	$= \forall x. fol_x(E) \leftrightarrow fol_x(F)$	$fol_x(K_1 \sqcap K_2)$	$= fol_x(K_1) \wedge fol_x(K_2)$
$fol(P \sqsubseteq Q)$	$= \forall x, y. P(x, y) \rightarrow Q(x, y)$	$fol_x(C_1 \sqcup C_2)$	$= fol_x(C_1) \vee fol_x(C_2)$
$fol(P \equiv Q)$	$= \forall x, y. P(x, y) \leftrightarrow Q(x, y)$	$fol_x(\exists P.C)$	$= \exists y. P(x, y) \wedge fol_y(C)$
$fol(P \sqsubseteq Q^-)$	$= \forall x, y. P(x, y) \leftrightarrow Q(y, x)$	$fol_x(\forall P.D)$	$= \forall y. P(x, y) \rightarrow fol_y(D)$
$fol(P \equiv P^-)$	$= \forall x, y. P(x, y) \leftrightarrow P(y, x)$	$fol_x(\forall P^-.D)$	$= \forall y. P(y, x) \rightarrow fol_y(D)$
$fol(P^+ \sqsubseteq P)$	$= \forall x, y, z. P(x, y) \wedge P(y, z) \rightarrow P(x, z)$	$fol_x(\exists P.\{O\})$	$= \exists y. P(x, y) \wedge y = O$
$fol(\top \sqsubseteq \forall P.C)$	$= \forall x. fol_x(\forall P.C)$		
$fol(\top \sqsubseteq \forall P^-.C)$	$= \forall x. fol_x(\forall P^-.C)$		

$$\overline{D} ::= A \mid D_1 \sqcap D_2 \mid \exists P.\{O\} \mid \forall P.D$$

In all previous cases,  $A$  is an atomic class,  $P, Q$  are *property (i.e. role) names* and  $O$  is an *individual name*. In addition, the **ABox**  $\mathcal{A}$  contains a sequence of definitions  $\mathcal{A}_1, \dots, \mathcal{A}_m$  of the form:

$$\overline{P(A, B)} \mid \overline{D(A)}$$

where  $P$  is a *property name*,  $D \in \mathcal{R}$ , and  $A, B$  are *individual names*.

The logic-based semantics of such fragment can be defined by using the encoding of DL into First Order Logic (FOL) (see Figure 1). Basically, the proposed subset of DL restricts the form of class descriptions in right and left hand sides of subclass and class equivalence definitions, and in individual assertions. Such restriction is required according to [Vol04] in order to be able to encode the cited fragment of DL into logic programming. Following [Vol04], the universal quantification is only allowed in the right hand sides of DL formulas, which corresponds in the encoding to the occurrence of the same quantifier in the left hand sides (i.e. heads) of rules. Union formulas are required to occur in the left hand sides of DL formulas, which corresponds in the encoding to the definition of two rules.

Let us see an example of an ontology  $\mathcal{O}_0$  (see Figure 2). The ontology  $\mathcal{O}_0$  describes *meta-data* in the **TBox** defining that the elements of *Man* and the elements of *Woman* are elements of *Person* (cases (1) and (2)); and the elements of *Manuscript* are either elements of *Paper* or elements of *Book* (case (4)). In addition, a *Writer* is a *Person* who is the *author\_of* a *Manuscript* (case (3)), and the class *Reviewed* contains the elements of *Manuscript reviewed by a Person* (case (6)). Moreover, the *XMLBook* class contains the elements of *Manuscript* which have as *topic* the value “XML” ((5)). The classes *Score* and *Topic* contain, respectively, the values of the properties *rating* and *topic* associated to *Manuscript* (cases (7) and (8)). The property *average\_rating* is a subproperty of *rating* (case (10)). The property *writes* is equivalent to *author\_of* (case (9)), and *authored\_by* is the inverse property of *author\_of* (case (11)). Finally, the property *author\_of*, and conversely, *reviewed\_by*, has as domain a *Person* and as range a *Manuscript* (cases (12)-(15)).

The **ABox** describes *data* about two elements of *Book*: “Data on the Web” and “XML in Scotland” and a *Paper*: “Growing XQuery”. It describes the *author\_of* and *authored\_by* relationships for the elements of *Book* and the *writes*

**Fig. 2.** An Example of Ontology

TBox	
(1) $Man \sqsubseteq Person$	(2) $Woman \sqsubseteq Person$
(3) $Person \sqcap \exists author\_of.Manuscript \sqsubseteq Writer$	(4) $Paper \sqcup Book \sqsubseteq Manuscript$
(5) $Book \sqcap \exists topic.\{“XML”\} \sqsubseteq XMLbook$	(6) $Manuscript \sqcap \exists reviewed\_by.Person \sqsubseteq Reviewed$
(7) $Manuscript \sqsubseteq \forall rating.Score$	(8) $Manuscript \sqsubseteq \forall topic.Topic$
(9) $author\_of \equiv writes$	(10) $average\_rating \sqsubseteq rating$
(11) $authored\_by \equiv author\_of^-$	(12) $\top \sqsubseteq \forall author\_of.Manuscript$
(13) $\top \sqsubseteq \forall author\_of^-.Person$	(14) $\top \sqsubseteq \forall reviewed\_by.Person$
(15) $\top \sqsubseteq \forall reviewed\_by^-.Manuscript$	
ABox	
(1) $Man(“Abiteboul”)$	(3) $Man(“Suciu”)$
(2) $Man(“Buneman”)$	(5) $Book(“XML in Scotland”)$
(4) $Book(“Data on the Web”)$	(7) $Person(“Anonymous”)$
(6) $Paper(“Growing XQuery”)$	(9) $authored\_by(“Data on the Web”, “Buneman”)$
(8) $author\_of(“Abiteboul”, “Data on the Web”)$	(11) $author\_of(“Buneman”, “XML in Scotland”)$
(10) $author\_of(“Suciu”, “Data on the Web”)$	(13) $reviewed\_by(“Data on the Web”, “Anonymous”)$
(12) $writes(“Simeon”, “Growing XQuery”)$	(15) $average\_rating(“Data on the Web”, “good”)$
(14) $reviewed\_by(“Growing XQuery”, “Almendros”)$	(17) $average\_rating(“Growing XQuery”, “good”)$
(16) $rating(“XML in Scotland”, “excellent”)$	(19) $topic(“Data on the Web”, “Web”)$
(18) $topic(“Data on the Web”, “XML”)$	
(20) $topic(“XML in Scotland”, “XML”)$	

relation for the elements of *Paper*. In addition, the elements of *Book* and *Paper* have been reviewed and rated, and they are described by means of a topic.

### 3 A Query Language based on DL

In this section we will define the query language based on DL. Such query language will introduce variables in DL formulas in order to express the values to be retrieved in the query result. In addition, our query language can handle conjunctions of DL formulas. We will use variable names starting with lower-case letters to distinguish them from non-variables. Now, assuming a set  $\mathcal{V}_c$  of variables for classes  $c, d, \dots$  and a set  $\mathcal{V}_p$  of variables for properties  $p, q, \dots$ , and a set  $\mathcal{V}_i$  of variables for individuals  $a, b, \dots$ , a query  $\phi$  against of an ontology  $\mathcal{O}$  is a conjunction  $\varphi_1, \dots, \varphi_n$  where each  $\varphi$  has the form:

Query Language	
$\varphi ::= C \sqsubseteq D \mid E \equiv F \mid P \sqsubseteq Q \mid P \equiv Q \mid P(A, B) \mid D(A)$	

where  $C \in \mathcal{L}^\mathcal{V}$ ,  $D \in \mathcal{R}^\mathcal{V}$ ,  $E, F \in \mathcal{E}^\mathcal{V}$ ,  $P, Q \in \mathcal{P}^\mathcal{V}$  and  $A, B \in \mathcal{I}^\mathcal{V}$ . In addition,  $\mathcal{E}^\mathcal{V}$  contains the set of formulas of the form:

$c$	$c \in \mathcal{V}_c$
$A$	atomic class
$E_1 \sqcap E_2$	$E_1, E_2 \in \mathcal{E}^\mathcal{V}$
$\exists P.\{O\}$	$P \in \mathcal{P}^\mathcal{V}, O \in \mathcal{I}^\mathcal{V}$
$\exists P^-. \{O\}$	$P \in \mathcal{P}^\mathcal{V}, O \in \mathcal{I}^\mathcal{V}$

$\mathcal{L}^\mathcal{V}$  contains the set of formulas of the form:

$c$	$c \in \mathcal{V}_c$
$A$	atomic class
$C_1 \sqcap C_2$	$C_1, C_2 \in \mathcal{L}^\mathcal{V}$
$\exists P.\{O\}$	$P \in \mathcal{P}^\mathcal{V}, O \in \mathcal{I}_\mathcal{V}$
$\exists P^-. \{O\}$	$P \in \mathcal{P}^\mathcal{V}, O \in \mathcal{I}_\mathcal{V}$
$C_1 \sqcup C_2$	$C_1, C_2 \in \mathcal{L}^\mathcal{V}$
$\exists P.C$	$C \in \mathcal{L}^\mathcal{V}, P \in \mathcal{P}^\mathcal{V}$
$\exists P^-.C$	$C \in \mathcal{L}^\mathcal{V}, P \in \mathcal{P}^\mathcal{V}$

$\mathcal{R}^\mathcal{V}$  contains the set of formulas of the form:

$c$	$c \in \mathcal{V}_c$
$A$	atomic class
$D_1 \sqcap D_2$	$D_1, D_2 \in \mathcal{R}^\mathcal{V}$
$\exists P.\{O\}$	$P \in \mathcal{P}^\mathcal{V}, O \in \mathcal{I}_\mathcal{V}$
$\exists P^-. \{O\}$	$P \in \mathcal{P}^\mathcal{V}, O \in \mathcal{I}_\mathcal{V}$
$\forall P.D$	$D \in \mathcal{R}^\mathcal{V}, P \in \mathcal{P}^\mathcal{V}$
$\forall P^-.D$	$D \in \mathcal{R}^\mathcal{V}, P \in \mathcal{P}^\mathcal{V}$

and finally,  $\mathcal{P}^\mathcal{V}$  contains property names and elements of  $\mathcal{V}_p$ , and  $\mathcal{I}^\mathcal{V}$  contains individual names and elements of  $\mathcal{V}_i$ .

As in the case of the data and meta-data definition language, the query language is restricted to a fragment of DL in order to be encoded in logic programming (i.e. in Prolog).

Assuming that variable names start with lower case letters, queries are formulas like *type("Growing XQuery")* whose meaning is “Find the type of *Growing XQuery*”. The query  $\exists \textit{Reviewed\_by}^-. \{ \textit{"Growing XQuery"} \}(\textit{reviewer})$  means “Retrieve the reviewers of “*Growing XQuery*”” and  $\exists \textit{Reviewed\_by}.\{ \textit{"Growing XQuery"} \}(\textit{manuscript})$  means “Retrieve the manuscripts in which “*Growing XQuery*” is a reviewer”. Meta-data can be retrieved by means of our query language as follows. For instance, using union and intersection operators we can retrieve *intersection*  $\equiv \textit{Book} \sqcap \textit{Reviewed}$  whose meaning is “Find the intersection of *Book* and *Reviewed*”. The query  $\exists \textit{Writes.range} \sqsubseteq \textit{domain}$  means “Find the domains and ranges of *Writes*”. Finally,  $\exists \textit{property.Person} \sqsubseteq \textit{Book}$  means “Find the properties from *Book* to *Person*”.

### 3.1 Inference System

Now, we would like to show an inference system, denoted by  $\vdash_{OI}$ , in order to provide semantics to our query language,  $\mathcal{O} \vdash_{OI} \alpha$  means that  $\alpha$  is deduced from the rules of Figure 3.

The rules from **Eq1** to **Eq2** handle inference about equivalence. For instance, **Eq1** infers equivalence by reflexivity and **Eq2** infers equivalence by transitivity. The rules from **Sub1** to **Sub13** handle inference about subclasses. Cases from **Sub3** to **Sub8** define new subclass relationships from the already defined subclass relationships and union and intersection operators. Cases from **Sub9** to **Sub12** define new subclass relationships for complex formulas. The rules **Type1** to **Type11** infer type relationships using subclass and equivalence relationships. The most relevant ones are the cases from **Type6** to **Type10** defining the meaning of complex formulas w.r.t. individuals. Finally, the rules **Prop1** to **Prop7** infer relationships about roles. The most relevant ones are the case **Prop5** about the inverse of a property and the case **Prop6** about a transitive relationship.

**Fig. 3.** Inference System  $\vdash_{OI}$

Rule Name	Inference
(Eq1)	$\vdash_{OI} E \equiv E$
(Eq2)	$E \equiv F, F \equiv G \vdash_{OI} E \equiv G$
(Sub1)	$\vdash_{OI} C \sqsubseteq C$
(Sub2)	$E \equiv F \vdash_{OI} E \sqsubseteq F, F \sqsubseteq E$
(Sub3)	$C \sqsubseteq D, D \sqsubseteq E \vdash_{OI} C \sqsubseteq E$
(Sub4)	$C \sqcup D \sqsubseteq E \vdash_{OI} C \sqsubseteq E, D \sqsubseteq E$
(Sub5)	$E \sqsubseteq C \sqcap D \vdash_{OI} E \sqsubseteq C, E \sqsubseteq D$
(Sub6)	$C_1 \sqcap C_2 \sqsubseteq D, E \sqsubseteq C_1 \vdash_{OI} E \sqcap C_2 \sqsubseteq D$
(Sub7)	$C_1 \sqcup C_2 \sqsubseteq D, E \sqsubseteq C_1 \vdash_{OI} E \sqcup C_2 \sqsubseteq D$
(Sub8)	$C \sqsubseteq D_1 \sqcap D_2, D_1 \sqsubseteq E \vdash_{OI} C \sqsubseteq E \sqcap D_2$
(Sub9)	$\exists P.\{O\} \sqsubseteq D, Q \sqsubseteq P \vdash_{OI} \exists Q.\{O\} \sqsubseteq D$
(Sub10)	$\exists P.C \sqsubseteq D, Q \sqsubseteq P \vdash_{OI} \exists Q.C \sqsubseteq D$
(Sub11)	$C \sqsubseteq \exists P.\{O\}, P \sqsubseteq Q \vdash_{OI} C \sqsubseteq \exists Q.\{O\}$
(Sub12)	$C \sqsubseteq \forall P.D, Q \sqsubseteq P \vdash_{OI} C \sqsubseteq \forall Q.D$
(Sub13)	$\vdash_{OI} C \sqsubseteq \top$
(Type1)	$C(A), C \equiv D \vdash_{OI} D(A)$
(Type2)	$C(A), C \sqsubseteq D \vdash_{OI} D(A)$
(Type3)	$C(A), D(A) \vdash_{OI} (C \sqcap D)(A)$
(Type4)	$(C \sqcap D)(A) \vdash_{OI} C(A), D(A)$
(Type5)	$C \sqcup D \sqsubseteq E, C(A) \vdash_{OI} E(A)$
(Type6)	$C(B), P(A, B) \vdash_{OI} (\exists P.C)(A)$
(Type7)	$P(A, O) \vdash_{OI} (\exists P.\{O\})(A)$
(Type8)	$\exists P.C \sqsubseteq D, P(A, B), C(B) \vdash_{OI} D(A)$
(Type9)	$\exists P.\{O\} \sqsubseteq D, P(A, O) \vdash_{OI} D(A)$
(Type10)	$C \sqsubseteq \forall P.D, P(A, B), C(A) \vdash_{OI} D(B)$
(Type11)	$\vdash_{OI} \top(A)$
(Prop1)	$\vdash_{OI} P \equiv P$
(Prop2)	$\vdash_{OI} P \sqsubseteq P$
(Prop3)	$P \equiv Q \vdash_{OI} P \sqsubseteq Q, Q \sqsubseteq P$
(Prop4)	$P \sqsubseteq Q, P(A, B) \vdash_{OI} Q(A, B)$
(Prop5)	$P \equiv Q^-, P(A, B) \vdash_{OI} Q(B, A)$
(Prop6)	$P^+ \sqsubseteq P, P(A, B), P(B, C) \vdash_{OI} P(A, C)$
(Prop7)	$C \sqsubseteq \exists P.\{O\}, C(A) \vdash_{OI} P(A, O)$

Our inference system is able to infer new information from a given ontology. For instance,  $\mathcal{O}_0 \vdash_{OI} \text{Reviewed}(\text{"Data on the Web"})$ , using the following **TBox** and **ABox** information:

---

$\text{Book}(\text{"Data on the Web"}).$   
 $\text{Person}(\text{"Anonymous"}).$   
 $\text{Book} \sqsubseteq \text{Manuscript}.$   
 $\text{reviewed\_by}(\text{"Data on the Web"}, \text{"Anonymous"}).$   
 $\text{Manuscript} \sqcap \exists \text{Reviewed\_by}.\text{Person} \sqsubseteq \text{Reviewed}.$

---

by means of the following reasoning:

---

$\vdash_{(\text{Type1})} \text{Manuscript}(\text{"Data on the Web"})$   
 $\vdash_{(\text{Type6})} \exists \text{Reviewed\_by}.\text{Person}(\text{"Data on the Web"})$   
 $\vdash_{(\text{Type3})} (\text{Manuscript} \sqcap \exists \text{Reviewed\_by}.\text{Person})(\text{"Data on the Web"})$   
 $\vdash_{(\text{Type2})} \text{Reviewed}(\text{"Data on the Web"})$

---

### 3.2 Limitation of the Inference System

Our inference system can be used for inferring consequences from a given ontology. The idea is to apply the rules up to a fix point is reached. In order to ensure a such fix point exists the inference system has been designed for reasoning with atomic classes (i.e. in the  $\vdash_{OI}$ :  $C$ ,  $D$  and  $E$  are atomic classes). Reasoning with complex formulas, infinite subclass relationships are generated. For instance, by means of the **Sub1** rule:  $\forall P.C \sqsubseteq \forall P.C, \forall P.\forall P.C \sqsubseteq \forall P.\forall P.C$ , etc. In addition,

we have designed the inference system in order to be implemented in logic programming, in particular, in Prolog. It forces to limit the inference capabilities of our system. The inference system only handles the user-defined DL complex formulas (i.e. those included in the **TBox**). For instance, we cannot infer *new* relations like  $C \sqsubseteq \forall P.D$  because it requires to check all the relations between individuals for  $P$ . The same can be said for  $\exists P.C \sqsubseteq D$ . We believe that it is not a serious drawback of our inference system and query language assuming that the user has to define such relationships as meta-data in the **TBox** in order to be handle in queries.

## 4 Encoding into Prolog

Now, we would like to show how to use Prolog in our framework. The role of Prolog is double. Firstly, we can encode any given ontology instance of the considered fragment into Prolog. Secondly, our inference system  $\vdash_{OI}$  can be encoded into Prolog by means of rules, in such a way that a certain class of Prolog goals can be used as query language.

### 4.1 Ontology Instance Encoding

The encoding of an ontology instance consists of Prolog facts of a predicate called *triple*, representing the RDF-triple based representation of OWL. In the case of the **TBox**:  $en(C \sqsubseteq D) = triple(en(C), rdfs : subClassOf, en(D))$ ;  $en(E \equiv F) = triple(en(E), owl : equivalentClass, en(F))$ ;  $en(P \sqsubseteq Q) = triple(en(P), rdfs : subPropertyOf, en(Q))$ ; and  $en(P \equiv Q) = triple(en(P), owl : equivalentProperty, en(Q))$ . In addition,  $en(C)$ ,  $en(D)$ ,  $en(E)$ ,  $en(F)$ ,  $en(P)$  and  $en(Q)$  represents the encoding of classes and properties in which class and property names  $C, P, \dots$  are translated as Prolog atoms  $c, d, \dots$ . The special case of  $\top$  is encoded as  $en(\top) = owl : thing$ . In addition, Prolog terms are used for representing complex DL formulas as follows:  $en(P^+) = trans(en(P))$ ;  $en(P^-) = inv(en(P))$ ;  $en(\forall P.C) = forall(en(P), en(C))$ ;  $en(\exists P.C) = exists(en(P), en(C))$ ;  $en(\exists P.-\{O\}) = hasvalue(en(P), en(O))$ ;  $en(C \sqcap D) = inter(en(C), en(D))$  and  $en(C \sqcup D) = union(en(C), en(D))$ . Finally, the elements of the **ABox** are also encoded as Prolog facts relating pairs of individuals by means of properties, and defining memberships to classes:  $en(P(A, B)) = triple(A, en(P), B)$  and  $en(D(A)) = triple(A, rdf : type, D)$ . In the case of the ontology  $\mathcal{O}_0$ , we will have:

---

```
triple(man,rdfs:subClassOf,person).
triple(woman,rdfs:subClassOf,person).
triple(inter(person,exists(author_of,manuscript)), rdfs:subClassOf,writer).
triple(union(paper,book),rdfs:subClassOf,manuscript).
triple(inter(book,exists(topic,"XML")),rdfs:subClassOf,xmlbook).
triple(inter(manuscript,exists(reviewed_by,person)), rdfs:subClassOf,reviewed).
triple(manuscript,rdfs:subClassOf,forall(rating,score)).
triple(manuscript,rdfs:subClassOf,forall(topic,topic)).
triple(author_of,owl:equivalentProperty,writes).
triple(authored_by,owl:equivalentProperty,inv(author_of)).
```

```

triple(average_rating,rdfs:subPropertyOf,rating).
triple(owl:thing,rdfs:subPropertyOf, forall(author_of, manuscript)).
triple(owl:thing,rdfs:subPropertyOf, forall(inv(author_of), person)).
triple(owl:thing, rdfs:subPropertyOf, forall(inv(reviewed_by,person)).
triple(owl:thing, rdfs:subPropertyOf, forall(inv(reviewed_by),manuscript)).
triple("Abiteboul",rdf:type,man).
triple("Buneman",rdf:type,man).
triple("Suciu",rdf:type,man).
triple("Data on the Web",rdf:type,book).
triple("XML in Scotland",rdf:type,book).
triple("Growing XQuery",rdf:type,paper).
triple("Anonymous",rdf:type,person).
triple("Abiteboul",author_of, "Data on the Web").
triple("Data on the Web",authored_by, "Buneman").
triple("Suciu",author_of, "Data on the Web").
triple("Buneman",author_of, "XML in Scotland").
triple("Simeon",writes, "Growing XQuery").
triple("Data on the Web",reviewed_by, "Anonymous").
triple("Growing XQuery",reviewed_by, "Almendros").
triple("Data on the Web",average_rating, "good").
triple("XML in Scotland",rating, "excellent").
triple("Growing XQuery",rating, "good").
triple("Data on the Web",topic, "XML").
triple("Data on the Web",topic, "Web").
triple("XML in Scotland",topic, "XML").

```

---

## 4.2 Encoding of the Inference System $\vdash_{OI}$

Now, the second element of the encoding consists of Prolog rules for encoding the  $\vdash_{OI}$  inference system. The set of rules can be found in Figure 4, where facts for predicates *class*, *property* and *individual* are defined for each atomic class, property and individual. Each inference rule is encoded by means of a rule, but in some cases by means of two rules (cases **Sub2**, **Sub4**, **Sub5**, **Type4** and **Prop3**). Let us remark that a Prolog interpreter loops by applying some of the rules (for instance, **Eq2**). It can be avoided by memorizing triples in the interpreter. Alternatively, a bottom-up approach can be considered.

## 4.3 Using Prolog as Query Language

In this section, we will show how to use Prolog as query language for OWL. Basically, each query  $\phi = \varphi_1, \dots, \varphi_n$  in our query language can be encoded as a Prolog goal  $? - \text{triple}(\text{en}(\varphi_1)), \dots, \text{triple}(\text{en}(\varphi_n))$  in which each element of  $\mathcal{V}_c$ ,  $\mathcal{V}_p$  and  $\mathcal{V}_i$  is encoded as a Prolog Variable. Now, we will show examples of queries against the ontology  $\mathcal{O}_0$  and the corresponding answers. Let us remark that in Prolog variables start with upper case letters.

**Query 1:** The first query we like to show is “Retrieve the authors of manuscripts”, which can be expressed in our query language as  $(\exists \text{Author\_of.Manuscript})(\text{author})$ . It can be encoded as:

$? - \text{triple}(\text{Author}, \text{rdf} : \text{type}, \text{exists}(\text{author\_of}, \text{manuscript})).$

Let us remark that our inference system is able to infer that a *Paper* and a *Book* is a *Manuscript* and therefore the above query retrieves all the manuscripts

Fig. 4. Encoding of the Inference System

Rule Name	Prolog Rules
(Eq1)	$\text{triple}(E, \text{owl} : \text{equivalentClass}, E) : \neg \text{class}(E).$
(Eq2)	$\text{triple}(E, \text{owl} : \text{equivalentClass}, G) : \neg \text{triple}(E, \text{owl} : \text{equivalentClass}, F),$ $\text{triple}(F, \text{owl} : \text{equivalentClass}, G).$
(Sub1)	$\text{triple}(C, \text{rdfs} : \text{subClassOf}, C) : \neg \text{class}(C).$
(Sub2-I)	$\text{triple}(E, \text{rdfs} : \text{subClassOf}, F) : \neg \text{triple}(E, \text{owl} : \text{equivalentClass}, F).$
(Sub2-II)	$\text{triple}(F, \text{rdfs} : \text{subClassOf}, E) : \neg \text{triple}(E, \text{owl} : \text{equivalentClass}, F).$
(Sub3)	$\text{triple}(C, \text{rdfs} : \text{subClassOf}, E) : \neg \text{triple}(C, \text{rdfs} : \text{subClassOf}, D),$ $\text{triple}(D, \text{rdfs} : \text{subClassOf}, E).$
(Sub4-I)	$\text{triple}(D, \text{rdfs} : \text{subClassOf}, E) : \neg \text{triple}(\text{union}(C, D), \text{rdfs} : \text{subClassOf}, E).$
(Sub4-II)	$\text{triple}(C, \text{rdfs} : \text{subClassOf}, E) : \neg \text{triple}(\text{union}(C, D), \text{rdfs} : \text{subClassOf}, E).$
(Sub5-I)	$\text{triple}(E, \text{rdfs} : \text{subClassOf}, C) : \neg \text{triple}(E, \text{rdfs} : \text{subClassOf}, \text{inter}(C, D)).$
(Sub5-II)	$\text{triple}(E, \text{rdfs} : \text{subClassOf}, D) : \neg \text{triple}(E, \text{rdfs} : \text{subClassOf}, \text{inter}(C, D)).$
(Sub6)	$\text{triple}(\text{inter}(E, C_2), \text{rdfs} : \text{subClassOf}, D) : \neg$ $\text{triple}(\text{inter}(C_1, C_2), \text{rdfs} : \text{subClassOf}, D),$ $\text{triple}(E, \text{rdfs} : \text{subClassOf}, C_1).$
(Sub7)	$\text{triple}(\text{union}(E, C_2), \text{rdfs} : \text{subClassOf}, D) : \neg$ $\text{triple}(\text{union}(C_1, C_2), \text{rdfs} : \text{subClassOf}, D),$ $\text{triple}(E, \text{rdfs} : \text{subClassOf}, C_1).$
(Sub8)	$\text{triple}(C, \text{rdfs} : \text{subClassOf}, \text{inter}(E, C_2)) : \neg$ $\text{triple}(C, \text{rdfs} : \text{subClassOf}, \text{inter}(D_1, D_2)),$ $\text{triple}(D_1, \text{rdfs} : \text{subClassOf}, E).$
(Sub9)	$\text{triple}(\text{hasvalue}(Q, O), \text{rdfs} : \text{subClassOf}, D) : \neg \text{triple}(Q, \text{owl} : \text{subPropertyOf}, P),$ $\text{triple}(\text{hasvalue}(P, O), \text{rdfs} : \text{subClassOf}, D).$
(Sub10)	$\text{triple}(\text{exists}(Q, C), \text{rdfs} : \text{subClassOf}, D) : \neg \text{triple}(Q, \text{owl} : \text{subPropertyOf}, P),$ $\text{triple}(\text{exists}(P, C), \text{rdfs} : \text{subClassOf}, D).$
(Sub11)	$\text{triple}(C, \text{rdfs} : \text{subClassOf}, \text{hasvalue}(Q, O)) : \neg \text{triple}(P, \text{owl} : \text{subPropertyOf}, Q),$ $\text{triple}(C, \text{rdfs} : \text{subClassOf}, \text{hasvalue}(P, O)).$
(Sub12)	$\text{triple}(C, \text{rdfs} : \text{subClassOf}, \text{forall}(Q, D)) : \neg \text{triple}(Q, \text{owl} : \text{subPropertyOf}, P),$ $\text{triple}(C, \text{rdfs} : \text{subClassOf}, \text{forall}(P, D)).$
(Sub14)	$\text{triple}(C, \text{owl} : \text{subClassOf}, \text{owl} : \text{thing}) : \neg \text{class}(C).$
(Type1)	$\text{triple}(A, \text{rdf} : \text{type}, D) : \neg \text{triple}(A, \text{rdf} : \text{type}, C),$ $\text{triple}(C, \text{owl} : \text{equivalentClass}, D).$
(Type2)	$\text{triple}(A, \text{rdf} : \text{type}, D) : \neg \text{triple}(C, \text{rdfs} : \text{subClassOf}, D), \text{triple}(A, \text{rdf} : \text{type}, C).$
(Type3)	$\text{triple}(A, \text{rdf} : \text{type}, \text{inter}(C, D)) : \neg \text{triple}(A, \text{rdf} : \text{type}, C), \text{triple}(A, \text{rdf} : \text{type}, D).$
(Type4-I)	$\text{triple}(A, \text{rdf} : \text{type}, C) : \neg \text{triple}(A, \text{rdf} : \text{type}, \text{inter}(C, D)).$
(Type4-II)	$\text{triple}(A, \text{rdf} : \text{type}, D) : \neg \text{triple}(A, \text{rdf} : \text{type}, \text{inter}(C, D)).$
(Type5)	$\text{triple}(A, \text{rdf} : \text{type}, E) : \neg \text{triple}(\text{union}(C, D), \text{owl} : \text{subClassOf}, E),$ $\text{triple}(A, \text{rdf} : \text{type}, C).$
(Type6)	$\text{triple}(A, \text{rdf} : \text{type}, \text{exists}(P, C)) : \neg \text{triple}(B, \text{rdf} : \text{type}, C), \text{triple}(A, P, B).$
(Type7)	$\text{triple}(A, \text{rdf} : \text{type}, \text{hasvalue}(P, O)) : \neg \text{triple}(A, P, O).$
(Type8)	$\text{triple}(A, \text{rdf} : \text{type}, D) : \neg \text{triple}(\text{exists}(P, C), \text{rdfs} : \text{subClassOf}, D),$ $\text{triple}(A, P, B), \text{triple}(B, \text{rdf} : \text{type}, C).$
(Type9)	$\text{triple}(A, \text{rdf} : \text{type}, D) : \neg \text{triple}(\text{hasvalue}(P, O), \text{rdfs} : \text{subClassOf}, D),$ $\text{triple}(A, P, O).$
(Type10)	$\text{triple}(B, \text{rdf} : \text{type}, D) : \neg \text{triple}(C, \text{rdfs} : \text{subClassOf}, \text{forall}(P, D)),$ $\text{triple}(A, P, O), \text{triple}(A, \text{rdf} : \text{type}, C).$
(Type11)	$\text{triple}(A, \text{rdf} : \text{type}, \text{owl} : \text{thing}) : \neg \text{individual}(I).$
(Prop1)	$\text{triple}(P, \text{owl} : \text{equivalentProperty}, P) : \neg \text{property}(P).$
(Prop2)	$\text{triple}(P, \text{owl} : \text{subPropertyOf}, P) : \neg \text{property}(P).$
(Prop3-I)	$\text{triple}(P, \text{owl} : \text{subPropertyOf}, Q) : \neg \text{triple}(P, \text{owl} : \text{equivalentProperty}, Q).$
(Prop3-II)	$\text{triple}(Q, \text{owl} : \text{subPropertyOf}, P) : \neg \text{triple}(P, \text{owl} : \text{equivalentProperty}, Q).$
(Prop4)	$\text{triple}(A, Q, B) : \neg \text{triple}(P, \text{rdfs} : \text{subPropertyOf}, Q), \text{triple}(A, P, B).$
(Prop5)	$\text{triple}(B, Q, A) : \neg \text{triple}(P, \text{owl} : \text{equivalentProperty}, \text{inv}(Q)), \text{triple}(A, P, B).$
(Prop6)	$\text{triple}(A, P, C) : \neg \text{triple}(\text{trans}(P), \text{rdfs} : \text{subPropertyOf}, P),$ $\text{triple}(A, P, B), \text{triple}(B, P, C).$
(Prop7)	$\text{triple}(A, P, O) : \neg \text{triple}(C, \text{rdfs} : \text{subClassOf}, \text{hasvalue}(P, O)),$ $\text{triple}(A, \text{rdf} : \text{type}, C).$

of the ontology  $\mathcal{O}_0$ . In addition, our inference system is able to infer that *author\_of* is a equivalent property to *writes* and therefore both cases are retrieved by the query language. In this case, the answers will be "Abiteboul", "Buneman", "Suciu", "Simeon", "Buneman".

**Query 2:** The second query we would like to show is “Retrieve the books of topic XML” which can be expressed as  $(Book \sqcap \exists Topic.\{“XML”\})(book)$ . Now, it can be expressed as:

$$\overline{? - triple(Book, rdf : type, inter(book, exists(topic, “XML”)))}.$$

However, given that the ontology already includes the class “XMLBook” we can express the same query in a more concise way as  $XMLBook(book)$  and therefore as:

$$\overline{? - triple(Book, rdf : type, xmlbook)}$$

**Query 3:** The third query we would like to show is “Retrieve the writers of reviewed manuscripts”. It can be expressed as  $(\exists Writes.Reviewed)(author)$ . In this case, the query can be expressed as:

$$\overline{? - triple(Author, rdf : type, exists(writes, reviewed))}.$$

**Query 4:** Now, the query “Retrieve the papers together with the reviewers”. This query can be expressed in our query language as  $Paper(paper), \exists Reviewed\_by^- . \{paper\}(reviewer)$ . In this case the Prolog encoding is:

$$\overline{? - triple(Paper, rdf : type, paper), triple(Reviewer, rdf : type, hasvalue(inv(reviewed\_by), Paper))}.$$

**Query 5:** Let us see an example of query for retrieving meta-data from the ontology. For instance  $\exists Reviewed\_by.range \sqsubseteq domain$  whose meaning is “Find the domains and ranges of Reviewed\_by” is encoded as:

$$\overline{? - triple(exists(reviewed\_by, Range), rdfs : subClassOf, Domain)}.$$

In this case the answers will be *Manuscript*, *Book*, *Paper* for *Domain* and *Man*, *Woman*, *Person* for *Range*.

## 5 Implementation in a Relational Database System

Finally, we would like to present how to implement our query language in a RDBMS. Our proposal is similar to the followed in the Minerva tool [ZML<sup>+</sup>06] with some differences. Basically, we will include the facts of *triple* in a relational table, but complex terms like *exists(P, C)* will be stored in a separate table. Therefore we will have two tables called *TRIPLES* and *COMPLEX*.

For instance the ontology  $\mathcal{O}_0$  can be stored as in Figure 5. In order to retrieve  $triple(inter(person, exists(author\_of, manuscript)), rdfs:subClassOf, writer)$  from the tables we have to join the following three records:  $(id1, rdfs:subClassOf, writer)$  from *TRIPLES*, and  $(id1, inter, person, id2)$ ,  $(id2, exists, author\_of, manuscript)$  from *COMPLEX*.



**Fig. 5.** Implementation with Tables

Table TRIPLE			Table COMPLEX			
Subject	Property	Object	Identifier	Term	Arg1	Arg2
man	<i>rdfs:subClassOf</i>	person	id1	inter	person	id2
woman	<i>rdfs:subClassOf</i>	person	id2	exists	author_of	manuscript
id1	<i>rdfs:subClassOf</i>	writer	id3	union	paper	book
id3	<i>rdfs:subClassOf</i>	manuscript	id4	inter	book	id5
id4	<i>rdfs:subClassOf</i>	xmlbook.	id5	exists	topic	"XML"
id6	<i>rdfs:subClassOf</i>	reviewed.	id6	inter	manuscript	id7
manuscript	<i>rdfs:subClassOf</i>	id8	id7	exists	reviewed_by	person
...			id8	forall	rating	score
authored_by	<i>owl:equivalentProperty</i>	id9	id9	inv	author_of	-
...						
"Abiteboul"	<i>rdf:type</i>	man				
"Buneman"	<i>rdf:type</i>	man				
..						

The differences with respect to the Minerva tool is that in our proposal complex formulas (exists, forall, etc) are handled as values of tables instead of table names, therefore we are able to retrieve DL complex formulas as result of queries. Similarly to the Minerva tool, we assume that the table TRIPLES include all the OWL triples inferred from the given ontology. It is reasonable given that usually ontologies do not change in time. In order to infer all the consequences from a given ontology instance we can apply the inference rules of Figure 3 up to a fix point is reached.

Finally, our query language can be easily encoded in SQL, for instance, the query  $(\exists \text{ Author\_of.Manuscript})(\text{author})$  can be encoded as follows:

```

SELECT Subject FROM TRIPLES, COMPLEX
WHERE Property=rdf:typeOf and
Object=Identifier and Term=exists
and Arg1=author_of and Arg2=manuscript

```

## 6 Conclusions and Future Work

In this paper we have proposed a query language for OWL based on Prolog. The query language is able to query about data and meta-data of a given ontology. We have studied the implementation of the query language in Prolog and in a RDBMS. As future work, we would like to extend our approach to richer fragments of DL and therefore of OWL. We believe that some extensions could be possible following the same technique here presented. We are now developing a prototype of our proposal using SWI-Prolog and MySQL. We hope the prototype will be available soon in the Web page of the group <http://www.ual.es/~jalmen>.

## References

- [ABE06] J. M. Almendros-Jiménez, A. Becerra-Terón, and Francisco J. Enciso-Baños. Magic sets for the XPath language. *Journal of Universal Computer Science*, 12(11):1651–1678, 2006.

- [ABE08] J. M. Almendros-Jiménez, A. Becerra-Terón, and Francisco J. Enciso-Baños. Querying XML documents in logic programming. *Journal of Theory and Practice of Logic Programming*, 8(3):323–361, 2008.
- [ABE09] J. M. Almendros-Jiménez, A. Becerra-Terón, and F. J. Enciso-Baños. Integrating XQuery and Logic Programming. In *Proceedings of the 17th International Conference on Applications of Declarative Programming and Knowledge Management, INAP'07 and 21th Workshop on (Constraint) Logic Programming, WLP'07*, pages 117–135, Heidelberg, Germany, 2009. Springer LNAI, 5437.
- [AKK07] I. Astrova, N. Korda, and A. Kalja. Storing OWL Ontologies in SQL Relational Databases. *International Journal of Electrical, Computer, and Systems Engineering*, 1(4), 2007.
- [Alm08] J. M. Almendros-Jiménez. An RDF Query Language based on Logic Programming. In *Proceedings of the 3rd Int'l Workshop on Automated Specification and Verification of Web Systems*. Electronic Notes on Theoretical Computer Science (200), 67–85, 2008.
- [Alm09a] J. M. Almendros-Jiménez. An Encoding of XQuery in Prolog. In *Procs of the Sixth International XML Database Symposium XSym'09, at VLDB'09*. Springer, LNCS, to appear, 2009.
- [Alm09b] J. M. Almendros-Jiménez. Ontology Querying and Reasoning with XQuery. In *Proceedings of the PLAN-X 2009: Programming Language Techniques for XML, An ACM SIGPLAN Workshop co-located with POPL 2009*. <http://db.ucsd.edu/planx2009/papers.html>, 2009.
- [BKvH02] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *The Semantic Web - ISWC 2002*, pages 54–68. LNCS 2342, Springer, 2002.
- [Bor96] Alex Borgida. On the relative expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.
- [dBLPF05] Jos de Bruijn, Rubén Lara, Axel Polleres, and Dieter Fensel. OWL DL vs. OWL flight: conceptual modeling and reasoning for the semantic Web. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 623–632, New York, NY, USA, 2005. ACM.
- [dLC06] Cristian Pérez de Laborda and Stefan Conrad. Bringing Relational Data into the Semantic Web using SPARQL and Relational OWL. In *Procs. of ICDEW'06*, page 55, Los Alamitos, CA, USA, 2006. IEEE Computer Society.
- [dMRA08] María del Mar Roldán-García and José Francisco Aldana-Montes. DBOWL: Towards a Scalable and Persistent OWL Reasoner. In *ICIW*, pages 174–179. IEEE Computer Society, 2008.
- [GHVD03] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. of the International Conference on World Wide Web*, pages 48–57, USA, 2003. ACM Press.
- [GMF<sup>+</sup>03] J.H. Gennari, M.A. Musen, R.W. Fergerson, W.E. Grosso, M. Crubézy, H. Eriksson, N.F. Noy, and S.W. Tu. The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.
- [HM01] Volker Haarslev and Ralf Möller. Racer system description. In *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*, pages 701–706, London, UK, 2001. Springer-Verlag.

- [HMS07] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *J. Autom. Reasoning*, 39(3):351–384, 2007.
- [HPSvH03] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: the making of a Web Ontology Language. *J. Web Sem.*, 1(1):7–26, 2003.
- [JCJ<sup>+</sup>07] D. Jeong, M. Choi, Y. Jeon, Y. Han, L.T. Yang, Y. Jeong, and S. Han. Persistent Storage System for Efficient Management of OWL Web Ontology. In *UIC 2007*, pages 1089–1097. LNCS 4611, Springer, 2007.
- [Jen08] Jena. Semantic Web Framework for Java. URL: <http://jena.sourceforge.net>, 2008.
- [KOM05] A. Kiryakov, D. Ognyanov, and D. Manov. OWLIM-A Pragmatic Semantic Repository for OWL. In *WISE 2005 Workshops*, pages 182–192. LNCS 3807, Springer, 2005.
- [MWL<sup>+</sup>08] L. Ma, C. Wang, J. Lu, F. Cao, Y. Pan, and Y. Yu. Effective and efficient semantic web data management over DB2. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1183–1194. ACM New York, NY, USA, 2008.
- [SPG<sup>+</sup>07] Evren Sirin, Bijan Parsia, Bernardo C. Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, June 2007.
- [TH06] D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.
- [Vol04] Raphael Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, Universität Fridericiana zu Karlsruhe, 2004.
- [W3C04a] W3C. OWL Ontology Web Language. Technical report, [www.w3.org](http://www.w3.org), 2004.
- [W3C04b] W3C. Resource Description Framework (RDF). Technical report, [www.w3.org](http://www.w3.org), 2004.
- [WED<sup>+</sup>08] Z. Wu, G. Eadon, S. Das, E.I. Chong, V. Kolovski, M. Annamalai, and J. Srinivasan. Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1239–1248, 2008.
- [ZML<sup>+</sup>06] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan. Minerva: A Scalable OWL Ontology Storage and Inference System. In *ASWC 2006*, pages 429–443. LNCS 4185, Springer, 2006.



# Obtaining accessible RIA UIs by combining RUX-Method and SAW

Marino Linaje, Adolfo Lozano-Tello, Juan Carlos Preciado, Roberto Rodríguez,  
Fernando Sánchez-Figueroa

QUERCUS Software Engineering Group  
Universidad de Extremadura  
Escuela Politécnica. Avda. Universidad s/n, 10071 – Cáceres  
{mlinaje, alozano, jcpreciado, rre, fernando}@unex.es

**Abstract.** Web 2.0 introduces important features from the User Interface (UI) perspective such as multimedia support or high levels of interactions, among others. These features are mainly being implemented using Rich Internet Applications (RIAs) technologies that are challenging the way in which the Web is being developed. The popularity of RIAs is witnessed by the flourishing and rapid spread of their implementation technologies e.g., AJAX, Flash, FLEX, OpenLaszlo or Silverlight. However, from the UI accessibility point of view, these technologies pose new challenges that the Web Accessibility Initiative of the W3C is trying to solve through the use of a standard specification for Accessible Rich Internet Applications (WAI-ARIA). Currently, the introduction of properties defined in WAI-ARIA is being done in an ad-hoc manner due to the lack of models, methodologies and tools to support the design of accessible RIA UIs. In this paper, we propose a semantic approach to deal with this modeling issue through the combined use of RUX-Method, a model-based method that allows building RIA UIs and SAW, a System for Accessibility to the Web through the specification in an ontology of different roles, properties and states of accessibility proposed by WAI-ARIA.

## 1 Introduction

During the last years, the complexity of tasks performed through Web applications has increased significantly. Modern Web applications support sophisticated interactions, client-side processing, asynchronous communications, multimedia handling, and more. In this context, the traditional Web architecture (i.e., HTTP-HTML) has shown its limits, and developers are switching to novel technologies known under the collective name of Rich Internet Applications (RIAs). RIAs combine the benefits of the Web distribution architecture with the UI interactivity and multimedia support of desktop environments.

From the UI point of view, RIAs expand traditional Web UI features, providing *Richer content types*: i.e., homogeneous multimedia content, like video and audio; *Richer controls*: i.e., many advanced widgets (e.g., accordion, modal window, etc.) that can be used to expand the possibilities of the standard HTML controls (e.g.,

textinput, combobox, etc.); *Richer UI temporal behaviors*: i.e., animations, transition effects, etc.; *Richer interactivity*: i.e., advanced interaction support through Web-extended events in the widgets (e.g., drag-and-drop or double-click). Finally, they allow *single-page UI*: while the traditional Web UI is Multi-page i.e., the page is refreshed at each user's interaction, RIAs can use also the Single-page paradigm, where the UI is composed by elements that can be individually loaded, displayed or refreshed according to the UI requirements.

This richness is one of the main reasons why more and more developers are trying to adapt their applications by replacing the old UI with a new one using RIAs. However, this richness introduces new accessibility challenges, e.g., drag-and-drop interaction events are not available to users that cannot use a pointer. Another situation is e.g., when we consider the single-page paradigm where part of the UI changes in response to user's actions or time- or event-based updates; in this case, the new content may not be available to those users who are blind.

To provide an accessible user experience to people with special needs, assistive technologies need to be able to interact with these new widgets, behaviors and UI paradigms. Although some RIA technologies provide support for WCAG [4] or Section 508 [22] accessibility guidelines, these guides were created for traditional Web UIs and they are not comprehensive enough for RIA UI accessibility specification.

WAI-ARIA [1] tries to solve this problem by defining how information about these issues can be specified to be used later by assistive technologies. At the moment, WAI-ARIA is focused in those RIA UIs based on W3C standards. However, from our knowledge the introduction of accessibility features provided by WAI-ARIA is currently being done in an ad-hoc and craft manner due to the lack of models and methodologies to support the RIA UI design [2].

This lack of methodologies for RIA UI design was just the origin of RUX-Method [3], a method for enriching the UI of Model-based Web Applications with RIA features. In [3] we did not consider accessibility issues. However, and even when the WAI-ARIA is still a working draft, in this paper we consider the inclusion of accessibility issues according to these specifications due to the many advantages that it has already introduced for accessible RIA UIs.

Accessibility can be analyzed from different points of view, such as normative, legislative and technological. The legislative context is essential but not sufficient. It is necessary to rely on regulations that help us to break existing obstacles [4].

From the technological point of view, there are several tools; some of them are thought for users and others are for designers; some are aimed at overcoming hardware barriers and others are designed for overcoming software or content barriers. This technological dispersion, this variety of manufacturers and this lack of an integrated tool hinder real accessibility.

This situation was just the origin of SAW, a System for Accessibility to the Web [5] that takes into account the final user as well as the designer. SAW integrates different tools in order to overcome the three main barriers to accessibility: hardware, browser and content. One of the main pillars of SAW is an ontology for describing the HTML elements, their relationships and those properties concerning accessibility (following WCAG [4]).

In this paper we describe the combination of both, RUX-Method and SAW, to obtain Accessible RIA UIs. In this work the ontology used in SAW has been substituted by that defined in WAI-ARIA, which is constituted by roles, states and properties that set out an abstract model for accessible UIs. The bridge between RUX-Method and SAW relies on the Component Library provided by the former and the ontology provided by the latter. The information provided by the ontology is used to enrich the RUX-Method components with attributes related to accessibility. These enriched components can be mapped to accessibility frameworks that use this information to provide alternative access solutions or simply can be rendered by Web browsers that will treat the information accordingly. Currently, there are several browsers and assistive technologies that give support to the WAI-ARIA draft, such as the latest versions of Firefox, IE, Opera or Jaws.

The rest of the paper is as follows. Sections 2 and 3 show an overview of SAW and RUX-Method respectively. In Section 4 the combination of SAW and RUX-Method is shown. Finally, section 5 summarizes the paper and presents several considerations about related works.

## 2 SAW overview

SAW is an ontology-based software suite that aims at providing accessibility to the Web. It was thought for traditional Web UIs. The software suite incorporates a special mouse that allows surfing the net to blind users by providing Braille cells to identify the elements being surfed. The suite applications and their relationships are shown in Figure 1.

SAW represents the different accessibility attributes of Web page elements in an ontology [6] named ONTOSAW. This own representation skeleton is used by the SAW software modules as a common source of information. One of these modules is the Web page editor EDITSAW, which allows the designer to carry out semantic annotations inside the HTML code, adding special accessibility elements. These annotations, together with the outline defined in ONTOSAW are used by the NAVISAW browser to show this information to the user. NAVISAW contains a speech synthesizer, voice recognition software and a special mouse (i.e., MOUSESAW). For the purpose of this paper the role of NAVISAW is not relevant, since we plan to use standard browsers that support the WAI-ARIA specification. Interested readers can see [5] for further information about NAVISAW.

ONTOSAW contains the basic structural elements that may appear on pages written in XHTML such as paragraphs, images, tables, links, objects, etc. ONTOSAW also takes into account the WAI's recommendations that determine the attributes of these elements to make them accessible. Besides, ONTOSAW incorporates other additional attributes that will allow NAVISAW to offer some special perceptual possibilities to blind people, such as linking each web page element to a voice file or a Braille file that describes the content or functionality of some elements. EDITSAW gets as entries the XHTML document and the ontology. It produces an XHTML document enriched with semantic annotations (step 2 in Figure 1). The annotations are carried out manually by the designer. An example of a *Table* element can be seen below Figure 1,

where the attributes ONTOSAW:resume, ONTOSAW:rows, ONTOSAW:columns, ONTOSAW:Braille and ONTOSAW:voice, with their corresponding values, have been incorporated.

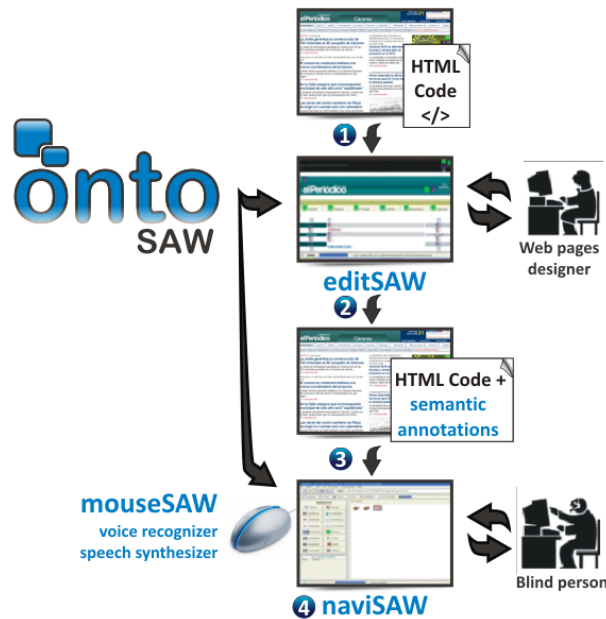


Fig. 1. SAW components and relationships

Simple example of annotated HTML page

```

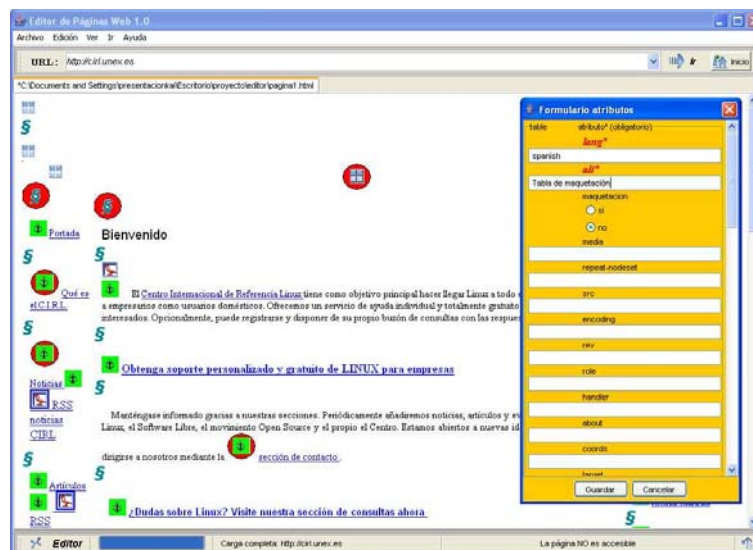
< table
  OntoSAW:resume="Price table of MP3 player"
  OntoSAW:rows=3   ontoSAW:columns=3
  OntoSAW:braille="http://quercusseg.unex.es/MP3player/table567/fictable567.bra"
  OntoSAW:voice="http://quercusseg.unex.es/MP3player/tabla567/fictable567.wav"
  width="100%" height="114" border="0">
  <tr> <td>Product</td> <td>Price</td> <td>Currency</td> </tr>
  <tr> <td>Player P1</td> <td>43</td> <td>Euro</td> </tr>
  <tr> <td>Player P2</td> <td>68</td> <td>Euro</td> </tr>
</table>

```

In order to make a Web page accessible, EDITSAW follows several steps. Firstly, EDITSAW connects to the ontology, usually situated in a common repository, through the Internet. This way, if the ontology has incorporated new elements and attributes according to the new recommendations from e.g., the WAI, these new characteristics will be considered dynamically by EDITSAW. After identifying the URL of the Web page that the designer wants to use, its contents are analyzed and the elements which are not accessible are shown (in Figure 2 showed with a circled icon). Then, the designer can modify the attributes of these elements (and others) simply by selecting the appropriate icon and filling in the values of the corresponding attributes (form at

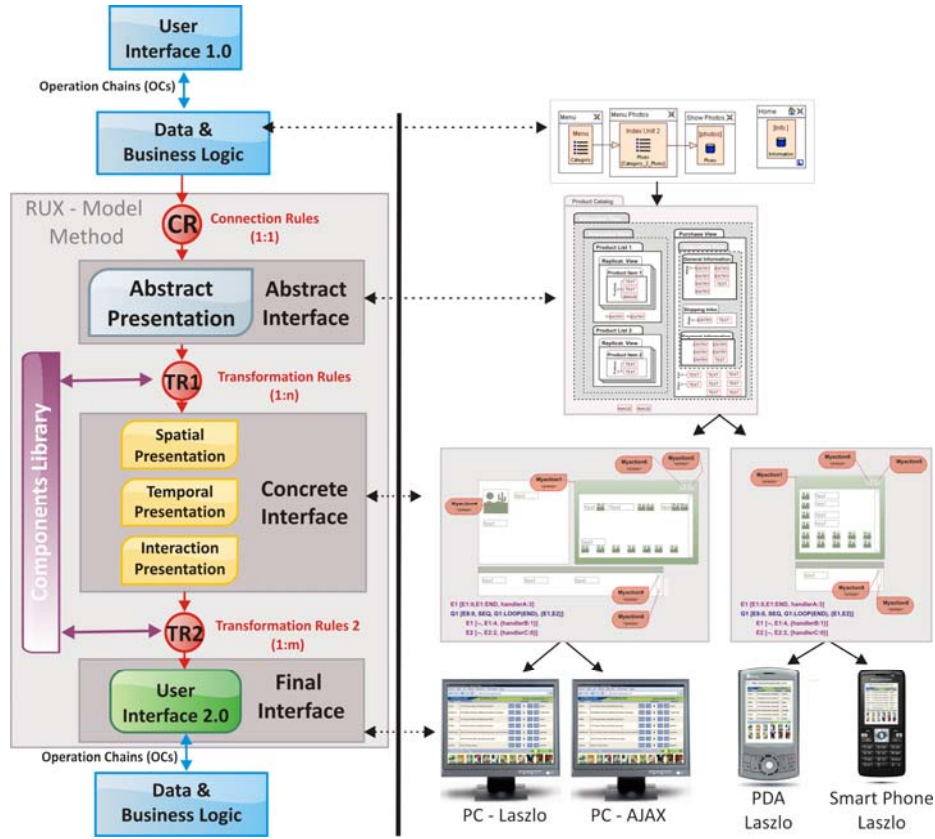


These special annotations that EDITSAW carries out form valid XHTML pages. Widely spread browsers ignore these labels although they show all the other attributes of the elements in a conventional way. On the other hand, NAVISAW will use these annotations to offer better accessibility to impaired users.

**Fig. 2.** Editing accessibility with EDITSAW

RUX-Method (Rich User eXperience Method, now on RUX) [7], is a model driven method which supports the design of multimedia, multi-modal and multi-device interactive Web 2.0 UIs for RIAs. RUX can be combined with many Web models such as WebML [8], UWE [9] OOHDm [10] or OO-H [11] which model the data and business logic in order to build complete rich Web Applications.

89



**Fig. 3.** RUX-Method architecture overview

In RUX, the Abstract UI provides a conceptual representation of the UI with all the features that are common to all the RIA devices and development platforms, without any kind of spatial, look&feel or behaviour dependencies. Following this idea, each component of the Abstract UI is also independent from any specific RIA device and rendering technology. This RUX UI level works with the following conceptual components: views (i.e., any different types of containers like alternative, hierarchical, etc.), media (i.e., any different type of media like text, video, images, etc.) and connectors (the connection with the business logic of the underlying Web model).

In the Concrete UI, we are able to optimize the UI for a specific device or a set of devices. Concrete UI is divided into three Presentation levels in order to provide a deeper separation of concerns according to the Web UIs requests: Spatial, Temporal and Interaction Presentation. Spatial Presentation allows the spatial arrangement of the UI to be specified, as well as the look&feel. Temporal Presentation allows the specification of those behaviours which require a temporal synchronization (e.g. animations). Finally, Interaction Presentation allows modelling the behaviours that the user produces through events over the UI.

The RUX process ends with the Final UI specification which provides the code generation of the modelled application when using RUX-Tool<sup>1</sup> [3]. This generated code is specific for a device or a set of devices and for a RIA development platform. This code is deployed together with data and business logic code generated using other CASE Tools (e.g., WebRatio).

There are two kinds of adaptation phases in RUX according to the UI levels defined above. Firstly, the adaptation phase that catches and adapts Web data, contents and navigation from the underlying Web model to RUX Abstract UI is called *Connection Rules*. Secondly, the adaptation phase that fits this Abstract UI to one or more particular devices and grants a right access to the business logic is called *Transformation Rules 1 (TR1)*.

Finally, there is an additional transformation phase supporting and ensuring the right code generation (*Transformation Rules 2 or TR2*).

Closely related to the Transformation Rules, is the Component Library (depicted in Figure 4). Each RUX UI Component specification is stored in this Library. The Library also stores how the transformations among components of different levels will be carried out.

The main interest for this paper relies on the Component Library, so next is briefly explained.

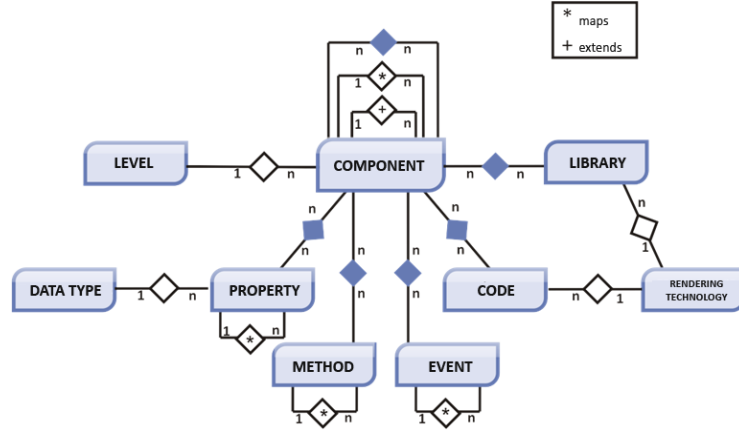
### 3.1 Component Library

The Component Library is responsible for: 1) storing the component specification (mainly composed of name, methods, properties and events), 2) specifying the transformation/mapping capabilities for each component from an UI level into other components in the following UI level and 3) keeping the hierarchy among components at each UI level independently from other levels. The set of UI Components that can be defined in the Library can be increased or modified by the designer according to the specifications of the project at any time during the UI design phase. In addition, the set of available transformations/adaptations for each Component can be also increased or updated according to the UI Components included in the Library. Summarizing, for a given component several transformations can be defined depending on the target components that can be *N*. E.g., an Abstract UI component is *Text* whose type can be input or output; this component could be transformed to the *RichTextEdit* or *TextControl* Concrete UI components or to any other component that the designer decides to integrate in the Library.

The Component Library is partially based on XICL (eXtensible user Interface Components Language) [13]. XICL is an extensible XML-based mark-up language for the development of UI components in Web applications. However, RUX extends XICL to define UI components (properties, methods and so on) and to establish mapping options among components of adjacent UI levels. Figure 4 illustrates the Entity-Relationship diagram for the Component Library.

---

<sup>1</sup> The RUX-Method CASE tool



**Fig. 4.** Component Library E-R description

#### 4. Combining editSAW and RUX using WAI-ARIA

Taking into account the satisfactory results of SAW, we decided to follow the same strategy for making accessible UIs based on RIA technologies. The idea consists in replacing ONTOSAW by WAI-ARIA ontology. If SAW project added accessibility information to HTML elements, in this case we need to add accessibility information to the description of RUX components at the level of the Concrete UI.

The incorporation of WAI-ARIA is a way to provide proper type semantics on custom widgets to make them accessible, usable and interoperable with assistive technologies [1]. This specification identifies the types of widgets and structures that are recognized by accessibility products, by providing an ontology of corresponding roles that can be attached to content. This allows elements with a given role to be understood as a particular widget or structural type regardless of any semantic inherited from the implementing technology. Roles are a common property of platform Accessibility APIs which applications use to support assistive technologies. Assistive technology can then use the role information to provide effective presentation and interaction with these elements.

The role taxonomy currently includes interaction widget (UI widget) and structural document (content organization) types of objects. The role taxonomy describes inheritance (widgets that are types of other widgets).

Changeable states and properties of elements are also defined in this specification. States and Properties are used to declare important properties of an element that affect and describe interaction. These properties enable the user agent or operating system to properly handle the element even when these properties are altered dynamically by

scripts. For example, alternative input and output technology such as screen readers must recognize if an object is disabled, checked, focused, collapsed, hidden, etc.

Next, we show as an example the *textBox* role of the WAI-ARIA ontology represented in OWL language. As shown in the code, the *textBox* class contains attributes that describe its accessibility: *autocomplete*, *multiline* and *readonly*. But also, other inherited attributes such as *describedby*, *hidden* or *required*, specified in the higher class named *roletype*. Inheritance occurs because *textBox* is a subclass of *input*, *input* is a subclass of *widget*, and *widget* is a subclass of a superclass named *roletype*.

Example of role *textBox* in WAI-ARIA ontology.

```
<owl:Class rdf:ID="textBox">
  <rdfs:subClassOf rdf:resource="#input"/>
  <rdfs:seeAlso rdf:resource="http://www.w3.org/TR/2007/
    REC-xforms-20071029/#ui-input"/>
  <rdfs:seeAlso rdf:resource="http://www.w3.org/TR/html4/
    interact/forms.html#edef-TEXTAREA"/>
  <role:supportedState
    rdf:resource="http://www.w3.org/2005/07/aaa#autocomplete"/>
  <role:supportedState
    rdf:resource="http://www.w3.org/2005/07/aaa#multiline"/>
  <role:supportedState
    rdf:resource="http://www.w3.org/2005/07/aaa#readonly"/>
</owl:Class>
```

Some attributes of *roletype* class inherited by *textBox* class.

```
<owl:Class rdf:ID="roletype">
  <role:supportedState
    rdf:resource="http://www.w3.org/2005/07/aaa#describedby"/>
  <role:supportedState
    rdf:resource="http://www.w3.org/2005/07/aaa#hidden"/>
  <role:supportedState
    rdf:resource="http://www.w3.org/2005/07/aaa#required"/>
</owl:Class>
```

Due to the versatility of EDITSAW, we replaced ontoSAW by WAI-ARIA ontology to get accessible RIA UIs. We needed to introduce a new property in the component description that is the role of the component (accordingly to the roles used by WAI-ARIA).

Next, we show an excerpt of the Spatial Presentation (at the Concrete UI level) including an instance of the *AutoCompleteTextControl* component which has been previously included in the Library. The component is identified by the unique key CTMI1, whose origin is a *Text* Media Input from the Abstract UI level (identified by the unique key ATMI1).

Example of a simple component at the Concrete UI level

```
<spatialPresentation>
  <structure>
    <part id="CTMI1" class="AutoCompleteTextControl" source="ATMI1"/>
  </structure>
</spatialPresentation>
```

```

</structure>
<properties id="Style1" target="AutoCompleteTextControl">
  <property name="vAlign">center</property>
  <property name="hAlign">center</property>
  <property name="width">50%</property>
  <property name="height">50%</property>
  <property name="role">textbox</property>
</properties>
</spatialPresentation>

```

The node *Properties* identifies a set of properties related with a specific component or type/class of components. In the example several properties for the spatial arrangement of CTMI1 can be observed. In addition, the property *role* specifies the role that this component play from the accessibility point of view (explained in next section). The introduction of this new property is the only change we had to perform in the Component Library for accessibility purposes. The rest of the work is done by SAW using WAI-ARIA and this property, as explained in next sections.

Now, the components belonging to an application at the level of the Concrete UI become one of the entries of EDITSAW that gets as second entry the WAI-ARIA ontology. For each component, EDITSAW identifies the role and, using the ontology, it forces the user to provide appropriate or recommended values for the accessibility attributes of that component (properties and states). Once all the components of the RIA have been enriched, then the Final UI can be generated. The outputs of EDITSAW are the components with information regarding accessibility. Figure 5 sketches this new scenario for EDITSAW.

Next, we show the *AutoCompleteTextControl* component after being enriched by editSAW annotations according to the WAI-ARIA TextBox role. In the example, we use for the implementation an auto-complete AJAX component which needs to include a library and a specific script code to work. This auto-complete component includes the “autocomplete”, “readonly” and “multiline” attributes plus “required”, “describedby” and “hidden” that are inherited properties from *Roletype*.

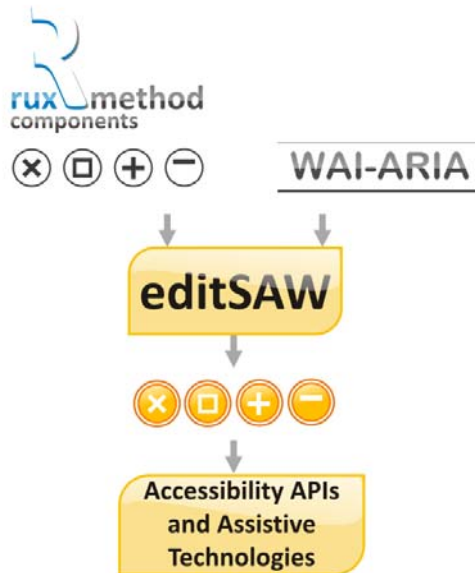
The implementation of the component (without accessibility issues) is stored in the Component Library. After filtered by EDITSAW, the component implementation is enriched giving the final code shown below.

Example of the *AutoCompleteTextControl* component enriched with attributes for accessibility

```

<input id="FTMI1" type="text" name="FTMI1"
class="AutoCompleteTextControl" role="TextBox" value=""
autocomplete="inline" readonly="false" multiline="false"
required="true" describedby="It is a textbox to fill the user
address" hidden="false"/>
<script type="text/javascript">
new Ajax.Autocompleter('FTMI1','update','page1.do?
source=DBConn1_store.data.items[0].data.mytext ', { tokens: ', ' } );
</script>

```



**Fig. 5.** Inputs and output of EDITSAW

The architecture of EDITSAW has been fully reused. The parser was implemented in Jena [14]. Instead of using NAVISAW for rendering the page, as done in SAW, now we can use whatever combination of browser and WAI-ARIA compliant Assistive Technology to make the interaction with RIA UIs accessible for people with special needs.

## 5 Conclusions and Related Work

This paper has introduced the combination of two previous works: RUX-Method and SAW to obtain accessible Rich Internet Applications. The result is a process that allows enriching already developed Model-Based Web applications with RIA features and accessibility issues. While RUX allows the introduction of a RIA UI in these applications, SAW gives support to the introduction of accessibility issues in those widgets provided by RUX. The join point between both works is the WAI-ARIA ontology. This ontology defines the attributes that must contain those widgets introduced by RIA technologies in order to be interpreted by the combination of accessibility APIs and assistive technologies.

Talking about related works is not an easy task due to several reasons:

- i) Currently, accessibility issues are not being taken into consideration by Web models, including those that are the most cited in the literature [7, 8, 9, 10]. Although some of these proposals have advanced approaches to consider RIA features, these ones are mainly related to aspects different

- from the UI design (logic or data distribution, synchronization, etc) and there is no known roadmap to include accessibility issues.
- ii) Those works addressing accessibility have a limited coverage and fall in one of the following fields:
- a. Works that are focused on the evaluation of the accessibility degree of Web pages such it is the case of Taw [15], Hera [16], or Wave [17]. These tools can be considered as post-implementation tools. A detailed explanation of tool support for accessibility assessment can be found in [18]. In this context EDITSAW can be seen as an evaluation tool but it goes a step further allowing the annotation of those inaccessible elements.
  - b. Works that are focused only on the UI design [19], paying no attention to the connection with other models providing data, business logic or communication issues which are very relevant in RIAs. RUX-Method also concentrates in the UI design, but provides appropriate mechanisms to connect to underlying Web models so applications with inaccessible Web 1.0 UIs developed with these models can be converted into accessible Web 2.0 UIs.
  - c. Works that indicate guidelines to include accessibility at the code level as it is the case of [4] or many other papers [20] or web pages [21] that one can find in the Web to make Ajax more accessible. In the context of RUX-Method these guidelines have been taken into account when generating the final code.

The conclusion is that, to our knowledge, there is no an integrated proposal able to include accessible RIA features in Web applications already developed using Web models and methodologies.

Although the obtained results are promising, we have only considered simplified application scenarios. Future work includes both, developing complex WAI-ARIA compliant RIAs and testing the results with visually impaired users (our group has an agreement with ONCE, the Spanish Organization for the Blind).

## Acknowledgments

This work has been partially supported by Spanish projects: TSI-020501-2008-47 (granted by Ministerio de Industria) and TIN2008-02985 (granted by Ministerio de Ciencia e Innovación) and by “Junta de Extremadura - Consejería de Infraestructuras y Desarrollo Tecnológico y el Fondo Europeo” (GRU09137)

## References

1. Accessible Rich Internet Applications. <http://www.w3.org/TR/wai-aria/>. 28-May-2009



2. Preciado, J.C., Linaje, M., Sánchez, F., Comai, S.: Necessity of methodologies to model rich Internet applications. In: Seventh IEEE International Symposium on Web Site Evolution, Budapest, 2005, pp. 7-13 (2005)
3. Linaje M., Preciado, J.C., Sánchez-Figueroa, F.: Enriching Model-Based Web Applications Presentation. *Journal of Web Engineering*, Vol. 7, Num. 3, pp 239-256 (2008)
4. Web Content Accessibility Guidelines. <http://www.w3.org/TR/WCAG20/>. 28-May-2009
5. Sánchez, F., Lozano, A., González, J., Macías, M.: SAW, a Set of Integrated Tools for Making the Web Accessible to Visually Impaired Users. *European Journal for the Informatics Professional, Upgrade*, Vol. VIII Núm. 2, Págs. 67-71 (2007)
6. Studer S., Benjamins R. y Fensel D.: Knowledge Engineering: Principles and Methods. *Data and Knowledge Engineering*, 25, 161-197 (1998)
7. Linaje, M., Preciado, J.C., Sánchez, F.: Engineering Rich Internet Application User Interfaces over legacy Web Models. *IEEE Internet Computing*, vol. 11, no. 6, pp. 53-59 (2007)
8. Brambilla, M., Comai, S., Fraternali, P., Matera, M.: Designing Web Applications with WebML and WebRatio, in *Web Engineering: Modelling and Implementing Web Applications*, G. Rossi, Ed. Springer, 2007
9. Koch, N.; Kraus, A., The expressive Power of UML-based Web Engineering, in *Second International Workshop on Web-oriented Software Technology (IWWOST02)*. CYTED, Málaga, 2002, pp. 105-119
10. Rossi, G.; Schwabe, D., The object-oriented hypermedia design model. *Communications of the ACM*, vol. 38, no. 8, pp. 45-46, 1995
11. Gómez, J., Cachero, C.: Extending UML to Model Web Interfaces. In *Information Modeling for Internet Applications*, P. Van Boomel, Ed. Idea Group Publishing, 2002, ch. 8, pp. 144-173
12. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez, V.: UsiXML: a Language Supporting Multi-Path Development of User Interfaces. *IFIP Working Conference on Engineering for HCI. LNCS*. 2005. 207-228
13. Gomes de sousa, L.J., Cavalcanti L., Using IMML and XICL components to develop multi-device web-based user interfaces. In *Brazilian symposium on Human factors in computing systems*. ACM, pp.138-147
14. Jena Web site: <http://jena.sourceforge.net/>. 28-May-2009
15. Web Accessibility Test tool. <http://www.tawdis.net>. 28-May-2009
16. Hera Accessibility Tool <http://www.sidar.org/hera/index.php.en>. 28-May-2009
17. Wave 3.0, Web Accessibility Tool. <http://wave.webaim.org>. 28-May-2009
18. Xiong, J., Winckler, M.: An Investigation of tool support for accessibility assessment throughout the development process of Web sites. In *Journal of Web Engineering*, Vol. 7, N° 4 pp. 281-298.
19. Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., Florins, M.: UsiXMLO: a User Interface Description Language for Specifying Multimodal User Interfaces. In *Proc. Of W3C Workshop on Multimodal Interaction WMI'2004*. Sophia Antipolis, 19-20 July 2004.
20. Kern, W. Web 2.0 – end of accessibility? – Analysis of most common problems with Web 2.0 based applications regarding Web accessibility. In *International Journal of Public Information Systems*, vol 2008:2
21. AJAX accessibility for Web sites: <http://www.webcredible.co.uk/user-friendly-resources/web-accessibility/ajax-accessibility.shtml>
22. Section 508 US Web site: <http://www.section508.gov/>. 28-May-2009



# Automatic Functional and Structural Test Case Generation for Web Applications based on Agile Frameworks

Boni García, Juan C. Dueñas, Hugo A. Parada G.

Departamento de Ingeniería de Sistemas Telemáticos  
ETSI Telecomunicación - Universidad Politécnica de Madrid  
Avda. Complutense s/n, 28040 Madrid, Spain  
{bgarcia, jcduenas, hparada}@dit.upm.es

**Abstract.** As web applications get widely available, more popular and sophisticated, there is an increasing need for methods and tools to produce them quickly. Web Application Frameworks (WAF) provide configurable libraries that can be extended with specific components. Testing is still the main technique to ensure the quality and the accomplishment of requirements. The approaches that emerged in the field of the automated test case generation are typically driven by specifications and models, rendering them useless when these are not available, such as in the case using WAF under agile processes. To overcome this difficulty, we propose mining the application-specific components to get information to feed test cases. This paper presents a method for the automation of test case generation for WAF-based applications developed under agile processes, considering domain elements as inputs for generation. This method is supported by an extensible testing platform named Automatic Testing Platform (ATP).

**Keywords:** Testing, Automation, Web, Framework, Agile, ATP.

## 1 Introduction

In the domain of Software Quality Engineering (SQE), software testing is an integral part of the Quality Assurance (QA) activities [1]. Testing is the process for discovering errors and failures in software systems; it is probably one of the most complex tasks in software development, since it is such a time-consuming and usually frustrating activity. Automated testing methods can significantly reduce the effort and cost of testing; but as regards the generation of tests and test data, it is a labor-intensive process yet.

Among the domain of applications with highest growth in the last years are web applications; there, the pressures to get products or services quickly forced developers to reduce the time and efforts devoted to the quality activities in general and testing in particular. Also, there have been lots of enhancements to practitioners' work, as the rapid pace of producing frameworks and even programming languages demonstrate: although Java is still the most important language in the server-side of enterprise web

applications in number of developments, new approaches emerged such as the Ruby on Rails, or the usage of frameworks such as are Grails, Trails, or Roma.

The need to produce results in a short time has even lead to a complete set of development methods under the “agile manifesto” umbrella. While these methods got success as regards pure development, it is now acknowledged the need for stronger QA activities, and some of the methods now incorporate testing activities as first-citizenship ones, such as in Test Driven Development (TDD) [2]. Automated testing is in place and many of housekeeping activities related to testing are then coped with. But, again, the problem is to generate tests (cases and data), which is regarded as a manual activity.

This paper presents our approach to the problem of automating test generation for the development of web applications built using WAF under agile processes. We have tried to put a bridge between WAF-based development and automated test generation, choosing the generation techniques already known which could be used in this particular domain, and developing a proof of concept for an automated test generation framework, the Automatic Testing Platform (ATP).

In the next section we revisit some state-of-the-art concepts that form the background of this paper. Section 3 provides the research statement, that is, the description of the problem we are dealing with in the specific context. Next section describes our proposed platform for automatic test case generation for agile Java frameworks, and section 5 introduces its usage and extension when using a specific WAF framework: the Romulus Framework. We conclude presenting our future work in section 6.

## **2 Background**

This section gives an indication of the current situation on the two main topics we are building upon: Web Application Frameworks (WAF), and automatic test case generation techniques. About WAF, we have focused on lightweight Java-based frameworks stemming from Java Enterprise Edition<sup>1</sup> (Java EE or JEE). On the other side, let us remind that automatic test case generation techniques are framed into the more general category of automatic testing; while there are lots of approaches and tools to support testing management, there are only a few for automatic test generation –we will focus on these–.

### **2.1 Web Applications Frameworks**

WAFs are defined as sets of classes that make up a reusable design for an application. Perhaps the most successful and widely adopted framework is Java EE, created by Sun Microsystems for developing server-side enterprise web applications using the Java language. However, in the last years real alternatives to Java EE in the domain of enterprise development emerged, in an attempt to ease the development. This is the

---

<sup>1</sup> <http://java.sun.com/javaee/>

case of the Spring Framework<sup>2</sup> [3], a full-stack layered Java-based WAF, based on code published in [4]. Since then, frameworks proliferated and specialized in tiers (persistence, web, flow, and so on); a complete classification can be found in [5]. On the other side, full stack WAFs boosted by the introduction of different programming languages, most of them based on dynamic scripting-based languages, and adherence to the agile principles of development. For example, Ruby On Rails [6] defines a new approach to web development, based on the principles of convention over configuration and avoiding redundancies (the DRY “don’t repeat yourself” principle), providing an agile web development framework that simplifies the development process and increases productivity for prototyping web applications.

However, Ruby on Rails is based on the Ruby language despite the fact that Java is the industry standard for business applications; forking development in two programming languages is simply not possible in this domain. Then it is necessary to bring the advantages provided by agile full-stack WAF, but at the same time reuse most libraries, subsystems and technologies already developed in Java, and retain the capability to interact with legacy Java systems. This caused the appearance of several Java-based alternatives to Ruby on Rails (collectively known as the “\*rails” frameworks), such as Grails, Trails or Roma, with different approaches to the reuse of previous frameworks and technologies and the application of agile principles to provide simplicity to web development.

Grails<sup>3</sup> [7] is a Java-based Rails-like development framework that provides Java integration while offering a dynamic oriented language, as it is based on the Groovy language [8], a dynamic object-oriented scripting language for the Java virtual machine with Java-like syntax. The Trails framework<sup>4</sup> allows rapid web application development creating pages from POJOs (Plain Old Java Objects); its stack is composed by String, Hibernate and Tapestry. It is based on the principles of Domain Driven Design (DDD) [9] and Model Driven Architecture (MDA) [10].

The Roma framework<sup>5</sup> has just introduced the idea of metaframework. It is also based on DDD using POJOs for application-specific functionality, but instead of providing a complete “wired” stack, it offers a common application programming interface to a set of pluggable Java frameworks such as Spring or JPOX (Java Persistent Objects) to transparently provide persistence, presentation or internationalization services. The goal there is getting POJO-based development with minimal coupling to the pluggable underlying frameworks. As regards testing, Roma does not offer facilities yet. Therefore, developers have to rely on third party non-integrated testing frameworks to perform testing activities, or do them by hand.

## 2.2 Automated Test Case Generation

As mentioned, testing enables software engineers to answer questions about the quality a software system. In essence, given a piece of software, testing consists of observing a sample of executions (test cases) of the system under test, and giving a

---

<sup>2</sup> <http://www.springframework.org/>

<sup>3</sup> <http://grails.org/>

<sup>4</sup> <http://www.trailsframework.org/>

<sup>5</sup> <http://www.romaframework.org/>

verdict over them. Managing the test cases, executing them on the proper part of the system under test, producing the required reports about errors found, among others, are the issues dealt with by automatic testing [11]. Part of automatic testing, and key in reducing the efforts and costs of testing, is automated test case generation: the techniques that help in generating the test cases, test data and oracles and verdicts that will feed the testing process. So far, these assets are generated by hand, as there is no information enough in the development process to start with; sometimes it is available but not represented using the proper modelling formalism.

Several approaches have been proposed in the literature for automated test case generation. The **specification-based** test case generation starts with system requirements expressed using a formal language such as SDL (Specification and Description Language). The Autolink [12] tool then generates test cases from the SDL specification and MSC (Message Sequence Chart) test purpose definitions. Other recent generators use specifications written using a programming language: JML (Java Modeling Language) is a language used to specify the behaviour of the code which can be used to generate unit tests to be executed by JUnit [13].

Other researchers have adopted a **model-based** test case generation, as they use modelling languages to get the specification and generate test cases. UML (Unified Modeling Language) diagrams such as state-charts, use-cases, sequence, and so on are widely employed in these solutions. For example, UMLTEST [14] is a test data generation tool integrated with Rational Rose<sup>6</sup> which employs UML state-chart to get the specification. Another example is TGV [15], a conformance test generator which creates test cases based on UML specifications and purposes.

In **path-oriented** test case generation, the control flow information is the input to identify a set of paths to be covered and generate the appropriate test cases. These techniques can be classified in static and dynamic ones. Static techniques rely on symbolic execution whereas dynamic techniques obtain the necessary data by executing the program under test. An example of dynamic path oriented test case generation is implemented by BINTTEST algorithm [16], which is employed for the test case generation based on binary search for the methods of a class.

**Random** test case generation techniques determine test cases on assumptions concerning fault distribution. Random testing can be seen as a second-class alternative to systematic testing. Systematic testing methods generate test cases only in the limiting sense that each domain point is a singleton sub-domain (also known as partitioning methods or Bounded Exhaustive Testing, BET). Random testing is literally the antithesis of systematic testing: no points are considered 'the same' and the sampling is over the entire input domain [17]. RUTE-J is a Java package providing tool support to programmers for randomized unit testing [18]. After that many tools have appeared with the same approach, like Jartege [19], in which random tests are generated and Korat [20], in which methods are tested in an exhaustive way by analysing the outputs obtained after executing all non-isomorphic tests cases.

In the **goal-oriented** test case generation approach, the cases are identified selecting goals such as a statement or branch or irrespective of the path taken [21]. Duy Cu Nguyen et al have developed a goal oriented testing methodology that takes design artefacts, specified as Tropos (an agent-oriented software development

---

<sup>6</sup> <http://www.ibm.com/software/rational/>

methodology) goal models, as central elements to derive test cases. The test suites are used to refine goal analysis and detect problems early at the requirement phase. They are executed afterwards to test the achievement of the goals from which they were derived [22].

**Intelligent** test case generation relies on complex computations to identify test cases. In the AETG system the tester first identifies parameters that define the space of possible test scenarios. Then the tester uses combinatorial designs to create a test plan that covers all pair-wise, triple or n-way combinations of test parameters. Heuristic algorithm has been developed to generate pair wise testing. Empirical results show that pair wise testing is practical and efficient for various types of software systems ([23,24]).

### 3 Research Statement

The focus of this paper is the generation of test cases for web applications based on Java agile WAF. These frameworks (Grails, Trails, or Roma) follow the agile approach, that is, rapid and easy development by simplifying processes and optimizing the return of investment and time to market. They are based on an Inversion of Control (IoC) container such as Spring and the development is oriented mainly to the domain (DDD), using POJOs for the implementation of the business logic. Operations such as CRUD (Create, Read, Update, and Delete) or the view generation are automated by the framework.

Current solutions, as proposed by TDD offer room for improvements. Agile developers are more interested in productivity and the easiness of development process, so we cannot expect they spend too much time in testing. Nevertheless, testing is still the most important activity for QA. For that reason, some testing tasks are required for enterprise web systems, even if they are based on agile frameworks.

Testing should be automated as far as possible; we propose a step further: tests should be generated automatically as far as possible. The methods we have just summarized present some limitations for our purposes: in specification-based testing, a prerequisite is to get a complete and consisted formal specification. This restriction is too strong in our context as no formal specifications are available. Model-based testing creates flexible, useful test automation from the model, but we cannot count on the model, because it is not even available. Random test case generation may create many test data, but might fail to find test case to satisfy requirements. In a path-oriented approach identifying the path might be infeasible or the test data generator might fail to find an input that will traverse the path. An intelligent approach generates test case quickly but is too complex.

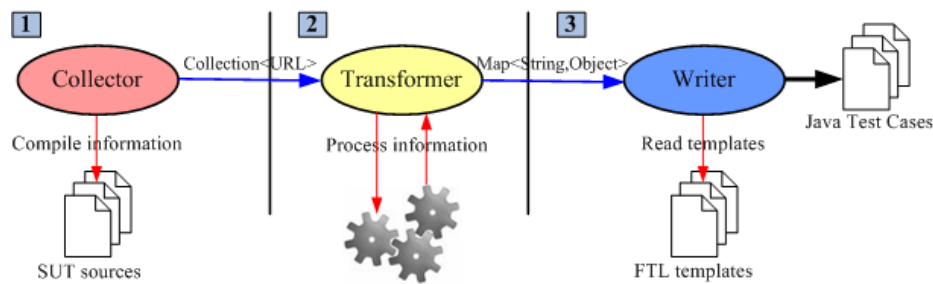
Due to the nature of an agile web application, we need a very flexible approach for the automatic test generation. We need to extract the information for generating the test cases from the source code of the System Under Test (SUT), or the information attached to the SUT. This requisite is not easy at all; in fact it is considered a dream for testing researchers [25]. Thus, we propose a test generation method based on the exploration of any piece of available information. “Generators” analyse the source code, annotations, configuration files, etc, and transform the information compiled to

test cases. These test cases are configurable with the aid of templates. This template-based approach offers a high degree of flexibility. We implement this method in the testing framework named Automatic Testing Platform (ATP).

## 4 Automatic Testing Platform (ATP)

ATP is based on entities called “generators” for the test case generation. These entities are in charge of gathering information, transform it and generate each single type of test cases. For achieving this goal we follow a three-tier architecture:

1. Collection of input data. This gathers information needed for the test case generation.
2. Transformation. This stage reads the information of the sources and prepares it for writing the test cases.
3. Test case generation. This stage generates the output, i.e., writes the test cases.



**Fig. 1.** Three-tier Testing Methodology.

The first stage is performed by entities named “collectors”. The aim of a collector is compiling the location of the input sources for the test cases. It examines the code looking for the sources in which the information is stored. The second stage is implemented by “transformers”. A transformer reads the proper information for a test case in the source found by collectors. This information is passed to the third stage, which is implemented by “writers”. The aim of a writer is to generate the test case, i.e., creating the test case file. Each writer is linked with a testing tool. The aggregation of these three entities (collector, transformer and writer) is known as a generator. This cascade process has the Java unit test case generation as a result. The way of working of a generator is implemented using the following snippet code:

```

public void generate() throws Exception {
    Collection<URL> in = (this.getCollector() != null) ?
        this.getCollector().collect() : null;
    Collection<Map<String, Object>> tc =
        this.getTransformer().transform(in);
    if (!tc.isEmpty()) {
        this.getWriter().write(tc);
    }
}

```



}

Given the structure of all the test cases is quite similar, we use templates. There are several Java open-source template engines available, such as Apache Velocity<sup>7</sup> and FreeMarker<sup>8</sup>. We have chosen FreeMarker because it has a better performance. The architecture of ATP<sup>9</sup> is completed with the usage of different pluggable open source testing frameworks, such as JUnit v3, JUnit v4, TestNG, Selenium, JUnitPerf, JMeter, and so on. These automate the execution of tests. In the next section we present in detail how it works for unit and system testing.

#### 4.1 Automatic Unit Test Case Generation

ATP provides a template-based platform for the automatic unit test generation for Java-based web applications. The tool is extensible and based on a plug-in design model, so the amount of generated test cases depends on the numbers of generators registered by ATP. As we know, a generator is linked to a specific unit technology because of its writer. The method for extending ATP by registering new unit generators in the platform is shown in Fig. 2.

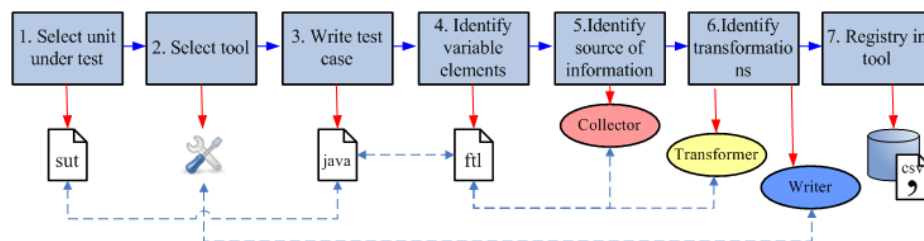


Fig. 2. Method for adding new generators in ATP.

1. Select unit under test. This first stage looks for the unit under test. We should focus on a layer of the web application architecture, and even more in a piece of this layer.
2. Select tool. The first version of ATP is focused on unit testing. This way, we select a unit testing Java framework supported by ATP: JUnit3, JUnit4, or TestNG.
3. Write test case. We have to write once the test case according to the selected tool. This part is performed by hand, and it will be the pattern of the test cases generated automatically by ATP.
4. Identify variable elements. The test case generated in the section before should be susceptible to be generalised. We have to identify the variable elements in the test case and change this parts for FreeMarker tags, e.g. \${element}. This stage produces the FreeMarker template, i.e., an FTL (FreeMarker Template Language) file.

<sup>7</sup> <http://velocity.apache.org/>

<sup>8</sup> <http://freemarker.sourceforge.net/>

<sup>9</sup> <http://sourceforge.net/projects/atestingp/>

5. Identify the source of information. According to the selected tags, we must be capable of locating the source of the information. This is handled by the collectors.
6. Identify transformation. Transformers must be able to pick up the information in the found sources by the collector and transform it to a map composed by pairs key-value. The set of keys must match the templates tags, and the value should be found in the sources found by the collector. The writer then creates the test cases using this map as input with the FTL template.
7. Registry in tool. When a generator (collector, transformer, and writer) is created, it must be registered in ATP, which is performed by adding its information to a CSV (Comma Separated Value) file which is processed by the ATP to register all the generators.

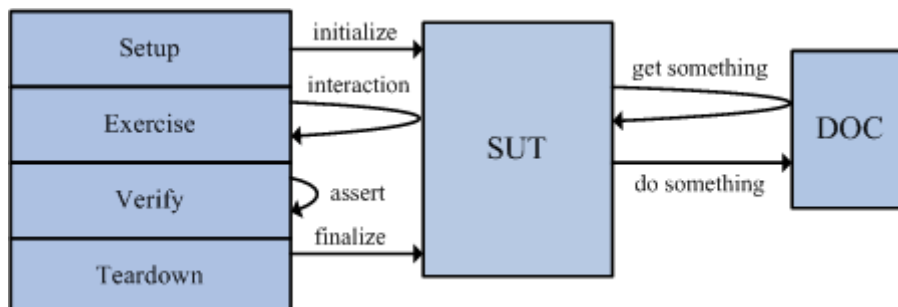
## 4.2 Semi-Automatic Unit Test Case Generation

In addition to the test case generation based on generators, ATP provides a module for the unit test cases generation based on the information stored in CSV files. For understanding how this module work, we first have to know what is a unit test case is.

A unit is the smallest testable part of an application. A unit test case strives for defect localization by ensuring a single condition test that exercise a single method or object in a single scenario. A general unit test case is composed by four phases [26]:

- Setup: Setting up the test fixture (the “before” situation) that is required for the System Under Test (SUT) to exhibit the expected behaviour as well as anything needed to put in place to be able to observe the actual outcome.
- Exercise: Interacting with SUT.
- Verify: Determining whether the expected outcome has been obtained (assertions).
- Teardown: Tearing down the test fixture to go back into the state in which SUT was.

SUT in short is whatever thing we are testing. Any part of the application or system we are building that is not included in the SUT may still be required to run the test (Depended-On Component, DOC). We call test fixture to everything we need in place to exercise the SUT. Both SUT and DOC are part of the test fixture. These concepts are illustrated in the next picture:



**Fig. 3.** Unit Test Case Structure.

For the generation of unit test cases, we need a way to codify the whole information needed. In other words, we need a way to express the four generic phases of a unit test case: setup, exercise, verify and teardown. Going deeper in this structure, we can analyse what exactly means each single phase in the Java language:

- Setup: Instantiating Java object and calling methods.
- Exercise: Calling Java methods.
- Verify: Assertions.
- Teardown: Calling Java methods.

In short, we can see there are three different sentences: instantiating Java objects, calling Java methods, and checking assertions (for each specific tool, such as JUnit or TestNG). We propose a way to codify these sentences in a simple way: by using CSV files as a test input. CSV files are basically plain text files in which the information is separated by tokens (typically the comma or the semicolon). Next paragraphs show the proposed CSV structure, and after that, we show some example to clarify the way of working of this module. Our main goal is to ease the developer the writing the information needed to create test cases, and CSV files can be written using widely available spreadsheets, or even plain text.

Each CSV file will be composed of one or more test cases. In the CSV a test case is a set of lines, that when it will be translated by the testing module of Roma/Romulus. It will generate a test case in JUnit3, JUnit4 or TestNG (depending of what indicated when using the module). The test cases in the CSV will be separated from each other by one or more blank lines. There are three types of CSV lines:

Instantiating Java object:

new, <class\_name>, param1, param2, ... , paramN (1)

Calling Java methods (two possibilities):

run, <class\_name>, param1, param2, ... , paramN (2)

run, \$<reference>, param1, param2, ... , paramN (3)

Assertions (two possibilities):

assert, \$<reference> (4)

assert, \$<reference1>, condition, \$<reference2> (5)

Let show an example of how the semi-automatic test generation module works. In this example, we have a web application with three classes, called Box, Store, and StaticStore. If a developer creates the following CSV file:

```

new,Store,1000
new,Box,3,2,2
run,$1,fits,$2
assert,$3

new,java.lang.Integer,1000
new,Store,$1
new,Box,3,2,2
run,$2,insert,$3
run,$2,getCapacity
assert,$5,!=,$1

run,StaticStore,getCapacity
assert,$1,<,1000

```

This CSV file (called testfile1.csv) will generate one Test Case JUnit3/JUnit4/TestNG (JUnit4 in the example below) file with three test case methods. The result is a complete test case able to be compiled and executed by the proper tool without any manual intervention.

```

package es.upm.dit.test.csv;

import org.junit.Test;
import org.junit.Assert;
import org.apache.commons.lang.builder.EqualsBuilder;
import es.upm.dit.test.StaticStore;
import es.upm.dit.test.domain.Store;
import es.upm.dit.test.domain.Box;
import java.lang.Integer;

public class TestCSVtestfile1 {
    @Test
    public void testCSV_1() throws Exception {
        Store store1 = new Store(1000);
        Box box2 = new Box(3,2,2);
        boolean boolean3 = store1.fits(box2);
        Assert.assertTrue(boolean3);
    }

    @Test
    public void testCSV_2() throws Exception {
        Integer integer1 = new Integer(1000);
        Store store2 = new Store(integer1);
        Box box3 = new Box(3,2,2);
        store2.insert(box3);
        Integer integer5 = store2.getCapacity();
        Assert.assertTrue(!EqualsBuilder.reflectionEquals(
            integer5, integer1));
    }

    @Test
    public void testCSV_3() throws Exception {
        int int1 = StaticStore.getCapacity();
    }
}

```

```

        Assert.assertTrue(int1 < 1000);
    }
}

```

### 4.3 Navigation Testing

Using the test generation strategy shown so far it is possible to test the internals of the web application built using WAF. Unit testing is an important task within QA, but it is not the only one. In fact, system testing could be as important as unit testing in web applications. For that reason, ATP contemplates system testing for web applications. More specifically, the testing technique covered is the automatically checking of the correctness of the navigations flows in a web application.

This approach is similar to the one proposed by model-based and path-oriented techniques described in section 2.2. Again, due to the agile nature of the SUT, we do not have the models or the specification where flows are described. Following the methodology proposed in section 4, we should extract the information about the navigation flow from the source code. This task is carried out by ATP. We provide generators for the automatic generation of this kind of functional test cases.

- Collector. This entity must find the source for the Finite State Machine (FSM) that implements the navigation flow. This task is specific for each framework. For example, in Spring Web Flow and Spring MVC (Model-View-Controller) the flow definitions are located in XML files. In the Roma framework this information is obtained by reading and transforming Java annotations found in DDD-POJOs.
- Transformer. This entity is responsible of translating the finite state machine (FSM) diagram to MBT<sup>10</sup>, which is an open source implementation of Model-based Testing [27,28] developed by Tigris. MBT allows generating test sequences from a finite-state machine in GraphML (XML-based file format for graphs). MBT can be instrumented in different ways, so the length, coverage, or state transversal on the FSM can be controlled.
- Writer. This entity is linked with the underlying testing framework. ATP employs Selenium for this kind of test cases. The information compiled for the transformer will be used for the writer in order to fill in the Selenium template. When acting on web applications, the points of control are those HTML elements that represent data input and interaction with the server, and the points of observation are the HTML elements that can be obtained as a response to an interaction. Selenium offers support to identify and handle these important elements.

### 4.4 Running and Reporting Test Cases

Other facility provided by ATP is the automatic generation of an Ant script for the execution and reporting of the cases. This script allows running all the test cases

---

<sup>10</sup> <http://mbt.tigris.org/>

generation, but it also generates HTML and XML reports with the results of the test case execution. In Fig. 4 we provide a screenshot of the generated HTML report:

Test	Methods Passed	Scenarios Passed	# skipped	# failed	Total Time	Included Groups	Excluded Groups
Ant test	41	41	0	5	2,5 seconds		

Class	Method	# of Scenarios	Time (Msecs)
Ant test — failed			
es.upm.dit.roma.test.domain.TestDomainPrimitiveStaticStore	testPrimitive	1	1
es.upm.dit.roma.test.domain.TestDomainPrivateStaticStore	testPrivateSet	1	1
es.upm.dit.roma.test.domain.TestDomainPublicGetStaticStore	testPublicGet	1	1
es.upm.dit.roma.test.domain.TestDomainPublicSetStaticStore	testPublicSet	1	1
es.upm.dit.roma.test.domain.TestDomainSetStaticStore	testSet	1	0
Ant test — passed			
es.upm.dit.roma.test.crud.TestCRUDBlog	testBlog	1	3
es.upm.dit.roma.test.crud.TestCRUDFilterBlog	testValidateBlogFilter	1	63
es.upm.dit.roma.test.crud.TestCRUDGeneratedBlog	testFilter	1	1
	testInstance	1	3
	testListable	1	0
	testMap	1	53
	testSecondary	1	1
es.upm.dit.roma.test.crud.TestCRUDInstanceBlog	testValidateBlogInstance	1	33
es.upm.dit.roma.test.crud.TestCRUDListableBlog	testValidateBlogInstance	1	31
es.upm.dit.roma.test.crud.TestCRUDPost	testCRUD	1	0
es.upm.dit.roma.test.domain.TestDomainGetBlog	testGet	1	1
es.upm.dit.roma.test.domain.TestDomainGetBox	testGet	1	1

Fig. 4. HTML report automatically generated.

## 5 Testing in Romulus Framework

The ATP tool has been extended to a specific existing agile WAF: the Romulus Framework, which is further based on the Roma Metaframework. This extension to ATP could be seen as a subclass of it, and therefore it inherits all the capabilities of the superclass. The name given to this extension is ATP4Romulus, and it can be found in the Romulus Framework website<sup>11</sup>.

As of today ATP4Romulus (v0.5) counts with 6 collectors, 19 transformers, and 4 writers. As a result, it has 57 registered generators. The following table summarizes the unit test cases generated with these 57 generators:

Table 1. Summary of the specific generator included in ATP4Romulus.

Aspect	Tool	Technique	Objective
Domain	JUnit v3-v4, TestNG	Structural	Keep the domain classes structure: accessors and mutators, private fields, and primitive fields
Domain	JUnit v3-v4, TestNG	Structural	Inheritance using the composite pattern
CRUD	JUnit v3-v4, TestNG	Functional	Ensuring Create, Read, Update and Delete
CRUD	JUnit v3-v4, TestNG	Structural	View-CRUD classes structure (Filter, Listable,

<sup>11</sup> <http://www.ict-romulus.eu/web/atp4romulus/>

View	TestNG			Instance and Repository)
	JUnit v3-v4, Structural			Layout and screen configuration
I18N	TestNG			
	JUnit v3-v4, Structural			Syntax and content of locale files
CSV	TestNG			
	JUnit v3-v4, Functional			Semi-automatic test case generation
Flow	TestNG			
	JUnit v3-v4, Functional			Navigation testing based on contract
Run and reporting	Ant	Functional	Running and reporting all the generated test cases	

---

A specific generator has been implemented for the navigation test cases. The collector looks for the Roma flow annotations. This information is used for the transformer to create the MBT file, and after that, the writer generates the test case for Selenium.

ATP4Romulus has been employed for the validation of the testing method proposed by this paper and implemented with the base platform ATP. As a result, we have obtained a tool that creates several test cases automatically, both functional and structural and in unit and system level. These tests can also be executed and reported automatically due to the fact that the script for launching these processes is also generated.

## 6 Conclusions and Future Work

As we know, testing is an essential activity in Software Quality Assurance. Enterprise applications have a technological heterogeneity and complexity that makes testing very difficult to put into practice. Moreover, a test process usually consists of several stages, such as test planning, design, coding and result analysis. As long as these activities are often performed manually, they are both time and cost consuming.

Enterprise web development with Java has been associated to Java EE for years. Nowadays, developers have real alternatives to Java EE, for example the Spring framework. Following the Ruby on Rails approach, new agile frameworks emerged in the Java world, such as Grails, Trails or Roma. When an application based on one of these frameworks reaches the production environment, these frameworks should provide testing facilities in order to accomplish functional and non-functional requirements.

This paper introduces the architecture of a system which tries to fully automate test generation for web applications based on agile frameworks: ATP (Automatic Testing Platform). This tool contains a collection of entities called generators, responsible of the automatic test case generation in several aspects: unit testing, system testing, test case execution and reporting. ATP is based on the usage of different pluggable testing tools: JUnit, TestNG and Selenium. ATP is also a template-based tool (FreeMarker), and therefore it is very flexible and scalable. This platform has been extending for the Romulus Framework in the ATP4Romulus platform.

These two platforms are currently in a continuous development cycle. The new capabilities of these tools will be in the integration, system level and performance testing. Thus, the platform will incorporate frameworks tools such as JMock or JMeter, in order to create new test cases for ensuring functional and non-functional requirements.

## Acknowledgements

This research project is funded by the European Commission under the R&D project ROMULUS (FP7-ICT-2007-1).

## References

1. Tian, J.: Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement. John Wiley & Sons (2005)
2. Ambler, S.: Agile Database Techniques: Effective Strategies for the Agile Software Developer. John Wiley & Sons (2003)
3. Walls, C.: Spring in Action. 2nd edition. Manning Publications (2007)
4. Johnson, R.: J2EE development frameworks. IEEE Computer Society Press (2005).
5. Shan, T.C., Hua, W.W.: Taxonomy of Java Web Application Frameworks. ICEBE '06: Proceedings of the IEEE International Conference on e-Business Engineering, Washington, DC, USA, IEEE Computer Society (2006)
6. Thomas, D., Heinemeier Hansson, D., Breedts, L., Clark, M., Duncan Davidson, J., Gehrtland, J., Schwarz, J.: Agile Web Development with Rails. 2nd edn. The Pragmatic Bookshelf (2006)
7. Rudolph, J.: Getting started with Grails. InfoQ – Enterprise Software Development Series (2007)
8. Koenig, D., Glover, A., King, P., Laforge, G., Skeet, J.: Groovy in Action. Manning publications Co. (2007)
9. Evans, E.: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison Wesley (2003)
10. Beydeda, S., Book, M., Gruhn, V.: Model-Driven Software Development. Springer (2005)
11. Farrell-Vinay, P.: Manage Software Testing. Auerbach Publications (2008)
12. Koch, B., Grabowski, J., Hogrefe, D., Schmitt, M.: Autolink, – A Tool for Automatic Test Generation from SDL Specifications. Workshop on Industrial Strength Formal Specification Techniques (WIFT). (1998)
13. Cheon, Y., Leavens, G.T.: The JML and JUnit way of unit testing and its implementation. European Conference on Object-Oriented Programming (ECOOP), Springer Berlin / Heidelberg (2002)
14. Offutt, J., Abdurazik, A.: Generating Tests from UML specifications. Second International Conference on the Unified Modeling Language, pp. 416-429, Fort Collins, CO. (1999)
15. Bousquet, L., Martin, H., Jezequel, J.M.: Conformance Testing from UML Specification Experience Report. Workshop of the pUML (2001)
16. Baydeda, S., Gruhn, V.: BINTEST – binary search-based test case generation. Computer Software and Applications Conference (COMPSAC), IEEE Computer Society Press (2003)
17. Hamlet, D.: When Only Random Testing Will Do. Proceedings of the First International Workshop on Random Testing (2006)



18. Andrews, J.H., Haldar, S., Lei, Y., Li, F.C.H.: Tool Support for Randomized Unit Testing. First International Workshop on Random Testing (2006)
19. Oriat, C.: Jartege: a tool for random generation of unit tests for Java classes. *Quality of Software Architectures and Software Quality*, Springer Berlin / Heidelberg (2005)
20. Boyapati, C., Khurshid, S., Marinov, D.: Korat: Automated testing based on Java predicates. *International Symposium on Software Testing and Analysis (ISSTA)*. (2002)
21. Pargas, R. P., Harrold, M. J., Peck, R.R.: Test-data generation using genetic algorithms. *Software Testing, Verification and Reliability*. (1999)
22. Nguyen D.C., Perini A., Tonella P.: *A Goal-Oriented Software Testing Methodology. Agent-Oriented Software Engineering VIII*. Springer (2008)
23. Cohen, D. M., Dalal, S. R., Fredman, M. L., Patton, G.C.: The AETG Design: an approach to testing based on Combinatorial design. *IEEE trans on Software Engineering* (1997)
24. Tracey, N., Clark, J., Mander, K.: Automated program flaw finding using simulated annealing. In *SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM Press (1998)
25. Bertolino, A.: Software testing research: Achivements, challenges, dreams. *Future of Software Engineering (FOSE)*. (2007)
26. Meszaros, G.: *xUnit Test Patterns. Refactoring Test Code*. Addison-Wesley (2008)
27. Baker, P., Dai, Z.D., Grabowski, J., Haugen, Ø., Schieferdecker, I., Williams, C.: *Model-Driven Testing Using the UML Testing Profile*. Springer (2008)
28. Utting, M., Legeard, B.: *Practical Model-Based Testing. A Tools Approach*. Elsevier (2007)



# Benchmarking and improving the quality of Norwegian municipality web sites

Morten Goodwin Olsen<sup>1</sup>, Annika Nietzio<sup>2</sup>, Mikael Snaprud<sup>1</sup>, and Frank Fardal<sup>3</sup>

<sup>1</sup> Tingtun AS,

PO Box 48, N-4791 Lillesand, Norway.

`morten.g.olsen@tingtun.no`, `mikael.snaprud@tingtun.no`

`http://www.tingtun.no`

<sup>2</sup> Forschungsinstitut Technologie und Behinderung (FTB)

der Evangelischen Stiftung Volmarstein, Grundschoetteler Str. 40

58300 Wetter (Ruhr), Germany.

`egovmon@ftb-net.de`

`http://www.ftb-net.de`

<sup>3</sup> Agency for Public Management and eGovernment (DIFI)

P.O. Box 8115 Dep, N-0032 Oslo, Norway

`Frank.Fardal@difi.no`

`http://www.difi.no`

**Abstract.** Automatic benchmarking can provide a reliable first insight into the accessibility status of a web site. The eGovMon project has developed a tool which can assess web sites according to a statistically sound sampling procedure. Additionally, the tool supports detailed evaluation of single web pages. This paper describes the process of data acquisition for the case of large scale accessibility benchmarking of Norwegian public web sites. An important contribution is the elaborated approach to communicate the results to the public web site owners which can help them to improve the quality of their web sites. An on-line interface enables them to perform evaluations of single web pages and receive immediate feedback. The close collaboration with the municipalities has lead to an overall higher quality both of Norwegian public web sites, the eGovMon tool and the underlying methodology.

Automated evaluation alone can not capture the whole picture, and should rather be seen as a complement to manual web accessibility evaluations. The Norwegian Agency for Public Management and eGovernment (DIFI/ Norge.no) carries out an annual web quality survey that includes manual assessment of web accessibility. We present a comparison between the Norge.no results and the data collected by the eGovMon tool and verify the statistical correlation.

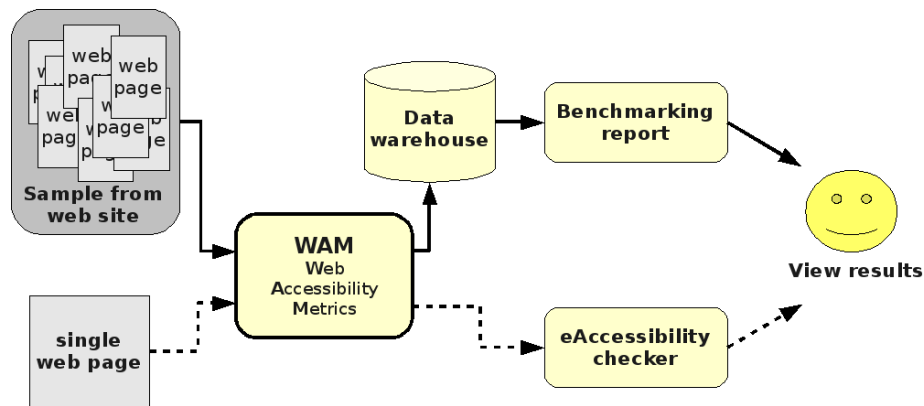
## 1 Introduction

The amount of information and the number of public services available on-line has been growing steadily over the past few years. Many administrative processes can be carried out via the Internet today.

However, a recent study [1] shows that there is still a large number of citizens who are not using eGovernment services. On the one hand the reasons are that the citizens feel no need to use eGovernment because they are not frequent users of the Internet in general or because they prefer to use other channels when interacting with public administration. On the other hand there are citizens who would like to use eGovernment applications but are prevented from doing so by poor accessibility and usability or because relevant content is missing or difficult to locate.

The latter group can benefit from improved quality and more user oriented design. In the long run this can help to increase take-up and facilitate the use also by the former group of citizens.

The Norwegian project eGovMon<sup>4</sup> is pursuing a two-fold strategy to advance accessibility and other attributes of web site quality.



**Fig. 1.** eGovMon System Architecture

*Large scale accessibility benchmarking.* The first part of the strategy consists of large scale accessibility evaluation. Frequently updated data on the accessibility status of Norwegian public web sites provides a bird's eye view of the situation and progress. Equal conditions under the evaluation ensure comparability of the results from different web sites and across time. This process is visualised by the solid arrows in Figure 1.

*On-line accessibility checker.* The second part of the strategy targets the web site maintainers on eye level. Detailed results for each web page are provided together with explanations and improvement suggestions. The pilot municipalities participating in the project have expressed a strong demand for practical

<sup>4</sup> The eGovMon project (<http://www.egovmon.no/>) is co-funded by the Research Council of Norway under the VERDIKT program. Project no.: Verdikt183392/S10

support. Often the web site maintainers are not aware of accessibility problems in their web sites or do not know how to resolve them. Therefore the eGovMon project tries to provide feedback and improvement suggestions that are easy to understand, and can facilitate the communication with technical staff, software vendors and web developers.

For this user group the eGovMon project has developed an easy to use on-line tool that can check single web pages. The checker provides detailed information on the identified accessibility barriers and suggests potential solutions. This application is shown by the dashed arrows in Figure 1.

*Collaboration with other initiatives.* In Norway, the Norge.no initiative of the Agency for Public Management and eGovernment (DIFI) carries out an annual systematic evaluation of the quality of Norwegian public web sites [2]. The criteria of this assessment address accessibility, usability and relevance of the provided content and services. The assessments are carried out manually by trained experts.

Automatic evaluations can support experts in their work, provide intermediary results between the annual evaluations, and support policy making and awareness raising. The eGovMon project is currently developing such automatic tools in collaboration with a group of 20 Norwegian municipalities, DIFI, and several additional government agencies and research partners from Norway and across Europe.

The remainder of this paper is organised as follows: Section 2 presents the eGovMon tool for automatic evaluation of accessibility. In Section 3 we explain the methodology used by the Norge.no evaluations. In Section 4 we compare the manually retrieved results from Norge.no with the automatically retrieved results from eGovMon. Finally, in Section 5, we present the eGovMon approach to communicating the results to Norwegian municipalities.

## 2 Automated evaluation of web accessibility in eGovMon

The large scale benchmarking approach applied in eGovMon is based on the Unified Web Evaluation Methodology (UWEM) version 1.2 [3], which was developed by the European Web Accessibility Benchmarking Cluster (WAB Cluster). The eGovMon system is an implementation of the *fully automated monitoring* application scenario described in UWEM.

*Methodology.* Web accessibility checking can be carried out in several ways along the same international standards. The evaluation methodologies used by evaluation and certification organisations in several European countries are different in subtle but meaningful ways [4], even though they are usually based on the Web Content Accessibility Guidelines 1.0 (WCAG 1.0) [5]. UWEM offers test descriptions to evaluate WCAG 1.0 conformance covering level AA, a clear sampling scheme, several reporting options, including score cards and other instruments

to help communicate the results of evaluations. UWEM was developed as the basis for web accessibility evaluation, policy support and possible certification in Europe [6].

## 2.1 Sampling of web pages

*Crawling.* The eGovMon system does not evaluate all web pages within a web site. Instead, it selects a random uniform sample from each web site. A random sample can only be drawn if the underlying population is known. Therefore each web site is explored by a web crawler trying to identify as many URLs as possible before the actual evaluation starts. The crawler follows a multithreaded breadth first search strategy and stores all discovered URLs in a URL database. The number of downloaded pages is constrained by the available download capacity. The URL discovery phase stops when 6000 web pages have been found.<sup>5</sup> In our experiments 85% of the web sites were crawled exhaustively. The remaining 15% of the sites are often considerably larger (sometimes consisting of up to several million single web pages).

In the next phase, 600 pages are randomly selected from the URL database, allowing the accessibility evaluation to be both representative of the web site as well as workable in practice.

*Choosing sample size.* There is a trade-off between system performance and accuracy of the results. Clearly, a large sample size would provide more precise results. The most accurate result could be achieved by evaluating every web page from the site. However, this is impossible in practice.

The sample size of 600 has been selected experimentally. Based on a number of test run the average standard deviation of the UWEM score within a web site could be estimated to  $\sigma = 0.25$ . Taking into account the potential values of the precision parameter  $d_1 = 0.05$  and  $d_2 = 0.02$  and the desired confidence intervals 95% (i.e.  $z_1 = 1.96$ ) or 99% (i.e.  $z_2 = 2.58$ ) we calculate the sample size as:

$$n = \frac{z^2 \sigma^2}{d^2}$$

In our setup, the evaluation speed is approximately 1.42 seconds per page, or 0.7 pages per second. The number of web sites which can be evaluated daily is therefore

$$N = \frac{1}{n} \cdot 0.7 \frac{\text{pages}}{\text{sec.}} \cdot 86400 \frac{\text{sec.}}{\text{day}}$$

This gives us the sample sizes presented in Table 1. A sample size of 600 pages per site allows the evaluation of approximately 100 web sites daily – an acceptable trade-off between precision and performance.

---

<sup>5</sup> For performance reasons eGovMon discovers URLs rather than download web pages. Discovering means detecting and finding any URL within the web site, and only downloading enough pages to detect 6000 URLs. For any web site with more than 6000 pages, there is a significant performance improvement gained by only detecting URLs compared to downloading.

$\sigma = 0.25$	$z_1 = 1.96$	$z_2 = 2.58$
$d_1 = 0.05$	$n \approx 96, N \approx 630$	$n \approx 166, N \approx 364$
$d_2 = 0.02$	$n \approx 600, N \approx 100$	$n \approx 1040, N \approx 58$

**Table 1.** Sample Size Calculations

## 2.2 Web accessibility testing

The system contains 23 web accessibility tests, which are all derived from the *fully automatable* tests in UWEM. The implementation is built on the Relaxed framework [7], which uses Java to parse the HTML source file into an HTML tree representation. Subsequently, this tree is assessed with a number of Schematron rules, which were developed specifically for UWEM.

Some restrictions apply when creating automatic measurements. Most significantly, many of the UWEM tests require human judgment. In fact only 26 of the 141 tests in UWEM are marked as automatable. As an example, automatic testing can find images without alternative text. However, to claim that an existing alternative text represents the corresponding image well, human judgment is needed. Thus, automatic evaluation can only be used to find barriers, not claim that a web site is accessible. Note that the automatic evaluation results can in some degree be used to outline to predict manual evaluation results [8].

*Schematron*. The Schematron language<sup>6</sup> is a rule-based validation language for making assertions about the presence or absence of patterns in XML trees. The context of a rule is described using XPath,<sup>7</sup> which provides a unique identification of all elements within the HTML tree. The test part of a Schematron rule consists of an XSLT statement that returns a boolean value. The rules are used to extract information about the different features of the HTML tree, e.g. presence or absence of attributes and elements or the relationship of parent, child, and sibling elements.

Schematron is only an intermediary step that provides direct access to the HTML structure. Other tests are then conducted based on the extracted data. This includes for instance operations such as matching strings with regular expressions or comparing to features that are not part of the HTML tree structure (i.e. information from the HTTP header or general information like the URL of the inspected page).

The Schematron approach provides an accurate and flexible implementation of the UWEM tests. However, problems can arise if the HTML source has severe parsing errors which make it impossible to construct the HTML tree in the first place. In these cases no accessibility evaluation can be carried out.

Initially we applied HTML Tidy<sup>8</sup> to clean up malformed HTML [9]. Pre-processing the HTML with HTML Tidy led to several issues. First of all, even

<sup>6</sup> <http://www.schematron.com>

<sup>7</sup> <http://www.w3.org/TR/xpath>

<sup>8</sup> <http://tidy.sourceforge.net/>

though the use of Tidy was set to minimum, it was not clear for the users how this changed the HTML. Users could claim that the HTML evaluated was not part of the web page but something adjusted with HTML Tidy. Additionally, using HTML Tidy made it difficult to identify the location of the barrier within the HTML source. However, even with the use of HTML Tidy, several web pages still could not be parsed. Because of these issues, the use of HTML Tidy was discarded in the final version of the implementation.<sup>9</sup>

*Results.* Each UWEM test is applied on each selected page. There are two possible outcomes: **fail** (barrier detected) and **pass** (no barrier detected). An example of a fail result is an image without an alternative description. This is a barrier because people who are unable to see images<sup>10</sup> rely on the alternative text to understand the image. When such alternative text is not present, the information conveyed in the image is lost to these users. All results are reported in the Evaluation and Report Language (EARL) [10].

*Storing results.* For each web site, the EARL output is incorporated into an RDF graph representing the web site. In addition to the accessibility results part of the EARL, the RDF graph contains information about web pages downloaded, HTTP header, language, technologies used. The data is stored in an RDF database which was developed specifically according to the project needs since the existing RDF database technologies were not able to provide sufficient speed for our application.

An Extract Transform Load (ETL) component reads the RDF-data, and stores the results in a data warehouse. The data warehouse [11] supports analysis with regards to multiple research questions. The outcome of the single tests is summarised into a web site accessibility score for the whole site. The score is calculated as the ratio of failed tests among to applied tests. The larger this ratio (percentage) of barriers detected, the less accessible the web site is. In a completely accessible web site there will not be any barriers and the percentage of detected barriers will be 0%. If half the tests detected barriers, the percentage of detected barriers would be 50%, and so on. In addition to these high level results, eGovMon presents also detailed results on page level.

### 3 Norge.no

The Norwegian Agency for Public Management and eGovernment (DIFI) conducts a yearly survey on the quality of Norwegian public web sites [2] – often called *Norge.no* after the web site where the results are published. The survey

---

<sup>9</sup> Web pages which could not be parsed are deliberately removed from the evaluation since we do not have any reliable results for these. Note that there is no direct indication that web pages we cannot parse are inaccessible.

<sup>10</sup> People who are unable to see images include for example people with visual impairments, people using mobile phones with images turned off to reduce data traffic, or people using web browsers without graphical user interface.



covers 34 indicators organised into three categories: accessibility, usability and relevance.

*Web Accessibility testing.* Twelve of the Norge.no indicators address accessibility. Out of these seven are directly related to WCAG 1.0 priority 1 and 2, three are related to WCAG 1.0 priority 3. The remaining two are not directly related to WCAG 1.0, but target other document formats such as PDF.

The evaluation is run in September and October each year and includes approximately 700 web sites at governmental and municipal level. The evaluations are carried out manually<sup>11</sup> by trained experts. On average, the review of one web site takes about one hour.

*Sampling and score.* Most tests are applied to two or three pages from the site. Sometimes the whole site is searched for certain features (e.g. data tables, documents in other formats). A failing test scores zero points. The maximum number of points for a test ranges from two up to five. The overall rating reports the percentage of the maximum number of points that has been achieved. These percentage values are then mapped to stars. The threshold values are based on a Gaussian distribution for one to five stars, with six stars as an “extra level” for exceptionally good web sites. The threshold values are presented in Table 2.

Stars	1 star	2 stars	3 stars	4 stars	5 stars	6 stars
Percentage	0 – 30%	31 – 42%	43 – 57%	58 – 69%	70 – 79%	80 – 100%

**Table 2.** Norge.no star rating

## 4 Results and comparison

Using the eGovMon tool, we evaluated the accessibility of web sites from 414 of 430 Norwegian municipalities in January 2009. The remaining municipalities had either no web site or the web site was not available during the evaluation<sup>12</sup>.

The eGovMon tool is based on WCAG 1.0 level AA and UWEM. In a web site not conforming to WCAG there will most likely be barriers preventing some users with disabilities from using the web site. It is worth noticing that people benefiting from accessible web sites are diverse and what may be barrier for one user may not be a barrier for others - even within the same disability group. Because of this, detecting and eliminating false positives would be very challenging. There has been some work trying to find false positive results in automatic

<sup>11</sup> Some tests rely on tool support, e.g. tests for sufficient colour contrast and validity of (X)HTML.

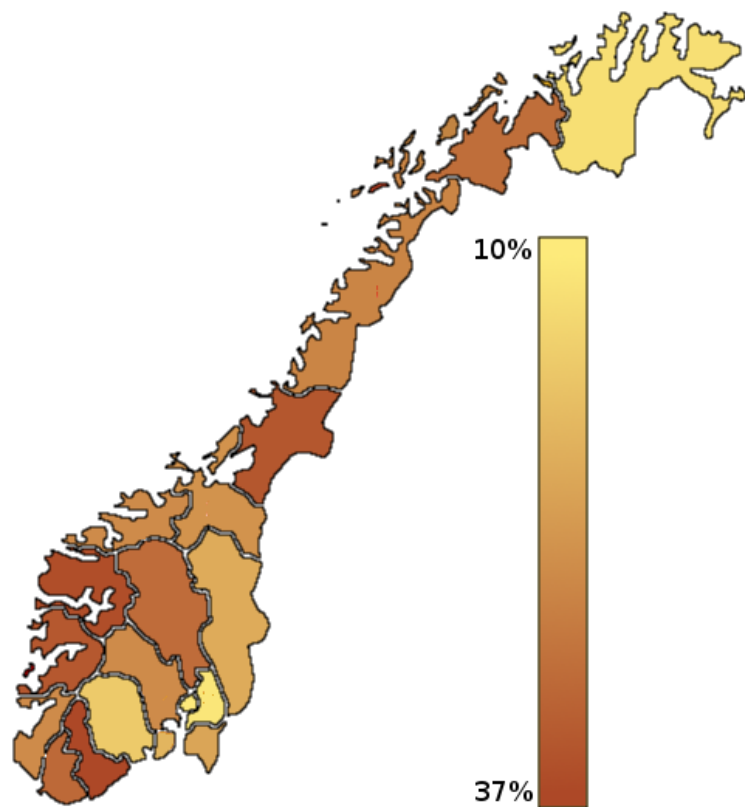
<sup>12</sup> Some municipality web sites deliberately prevent access from tools which are not known with the help of the robots exclusion standard.

accessibility measurements [12]. However, to the best of our knowledge, no such work has been carried out for UWEM.

#### 4.1 Accessibility of Norwegian Municipality Web Sites

The eGovMon results for single web sites indicate the percentage of barriers. While the results on county level are averages of the municipality web site results within the county. The accessibility results on county level are presented as a map of Norway in Figure 2. The results in this evaluation range from 10% to 37% of barriers detected by the tests. The darker colour means more barriers detected, while a lighter colour means less barriers detected. The county with the fewest detected barriers is the Norwegian capital Oslo. Even here, 10% of the eGovMon tests detected barriers. This shows that the public Norwegian web sites are far from being accessible. Additionally, our findings show that some barriers are more common than others.

1. **Invalid or deprecated (X)HTML and/or CSS** was the most common barrier found by the eGovMon tool and occurred in 99% of the evaluated web pages. (X)HTML and CSS are the most used technologies for web pages. The most recent version of these technologies are built with accessibility in mind, which means assistive technologies can more easily and successfully present the web page content when the latest (X)HTML and/or CSS are used correctly.
2. **Links with the same title but different target** occurred in 31% of the evaluated pages. Often links do not describe the target pages well. A typical example is having links with the text “read more”, which does not explain anything about the target page. Links should be more descriptive such as “read more about the economic crisis” or only “the economic crisis”. For fast and efficient navigation, some assistive technologies present all links within a web page to the user. However, if all links have the same text such as “read more”, this is not helpful.
3. **Graphical elements without textual alternative** were detected in 24% of the evaluated pages. The most common example of this is the use of images without alternative text, which causes problems for people with visual impairments who are unable to see the pictures. Any information conveyed in an image is lost to these users whenever a textual alternative is missing.
4. **Form elements without labels** occurred in 24% of the evaluated pages. An example of misuse would be not to correctly mark a search button as “search”. The fact that the web site is searchable, is sometimes understood by the context around the search field, such as a magnifying glass nearby. People with visual impairments and dyslexia sometimes have the web page text read out loud using screen readers, and may be unable to see the corresponding magnifying glass. If a text field is not clearly marked, it is challenging to know that it is intended for searching the web site.
5. **Mouse required** occurred in 11% of the evaluated pages. Web sites requiring the use of a mouse cause problems for people with motor impairments



**Fig. 2.** Map of Norway showing the eGovMon accessibility results from January 2009. A darker colour means more accessibility barriers found.

who often have challenges using such devices. An example is web sites with menu items which can only be accessed by clicking with a mouse but not by keyboard. Often, people with motor impairment are not able to use such web sites at all.

## 4.2 Results from eGovMon compared to International Surveys

None of the evaluated sites passed all eGovMon tests. This result is not in correlation with web accessibility surveys existing in the literature.

The study *Assessment of the Status of eAccessibility in Europe (MeAC)* [13], which has received much attention since it was published, shows that 12.5% of the web sites passed all automatic accessibility tests, and 5.3% passed all manual accessibility tests. Additionally, the United Nations Global Audit on Web Accessibility [14] indicates that 3% of the evaluated web sites pass all accessibility tests. It should be noted that the eGovMon survey only includes results from Norwegian web sites, whereas both the surveys from MeAC and United Nations evaluated web sites from the EU member states and United Nation member countries respectively. The eGovMon evaluation shows that none of the evaluated sites pass, which is clearly worse than web accessibility surveys results presented above. This discrepancy may be explained by the fact that eGovMon evaluates according to WCAG 1.0 level AA while both MeAC and United Nation survey only included results from WCAG 1.0 level A. Additionally, eGovMon evaluated up to 600 from each site while MeAC has included only 25. A more detailed comparison can be found in [15].

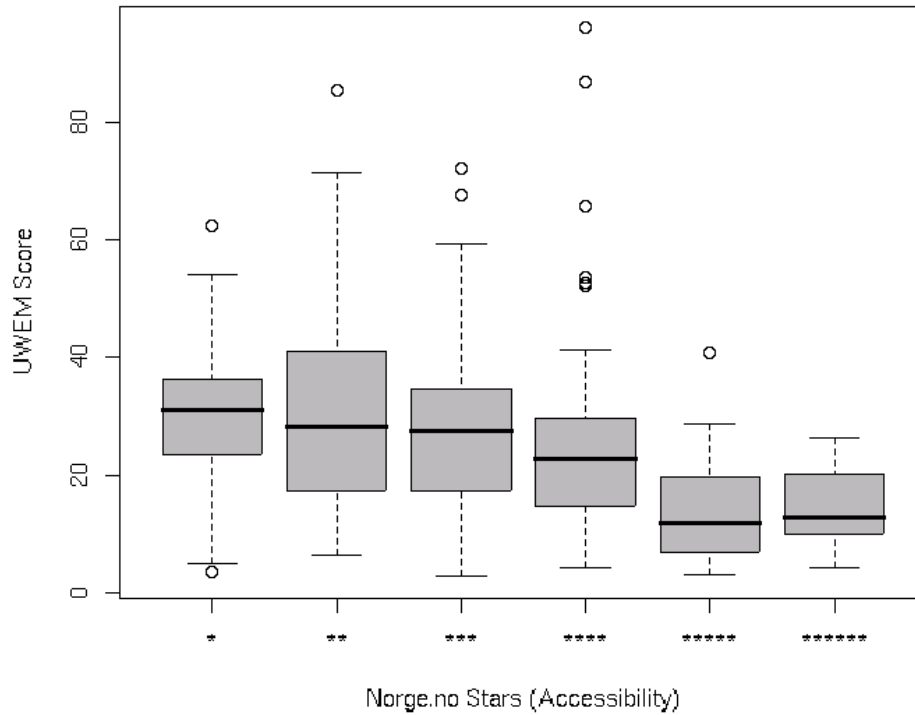
## 4.3 Results from eGovMon compared to National Survey

We are compared the result of eGovMon to the DIFI / Norge.no survey. This comparison includes only the accessibility part of the Norge.no survey. The results from Norge.no have been discretized into six levels using the predefined threshold values shown in Table 2. The more stars a web site has received the more accessible it is.

The UWEM score is not defined to match any distribution. Instead it presents the percentage of barriers detected within the applied tests. A low value means that few barriers were detected which indicates that the corresponding web site is accessible, while a high value indicates that the corresponding web site is inaccessible.

The two methodologies have few similarities. The Norge.no evaluations are conducted manually by trained experts, sometimes supported by tools, while the current eGovMon runs completely automatically. Furthermore, there exists only one test which is identical for both Norge.no and eGovMon (valid (X)HTML), and only two tests which are partially overlapping.

Furthermore, the Norge.no evaluations were carried out in September/October 2008, while the eGovMon evaluations were carried out in the beginning of January 2009. It is expected that several of the web sites have been updated in this period which will cause some inconsistency in the data.



**Fig. 3.** Comparison between accessibility results from Norge.no and eGovMon

Figure 3 presents the correlation between the expert evaluation results from Norge.no and the automatically retrieved results from eGovMon. Despite the methodological differences, the figure shows that there exists a correlation between these two evaluations. This shows that there is a solid dependency between the results which indicates that both methodologies are measuring accessibility. Web sites which perform good in one survey are very likely to get a similar result by the other (and vice versa).

We can clearly see that the average eGovMon web site score is better (fewer barriers detected) the more stars appointed to the web site by Norge.no. This is true for all groups of stars except for the web sites which received six stars and have been categorised as exceptionally good by Norge.no. These web sites receive a slightly worse score from eGovMon than the web sites which received five stars. This indicates that identification of good accessibility (six stars) cannot be done by automatic evaluation alone, but needs to be supported by manual assessment.

In addition, Figure 3 shows that of the 414 evaluated web sites there are twelve outliers. In ten of these sites, the eGovMon tool detected a large amount of deprecated elements and attributes, which in eGovMon and UWEM has a significant impact on the web site results. In contrast, deprecated elements or

attributes are not part of the evaluations of Norge.no. The remaining two outliers received a very good score by eGovMon while they only got one star by Norge.no. The reason for this discrepancy is not known, but the two web sites could have been updated between the Norge.no and eGovMon evaluation.

## 5 Communication of results

The results of benchmarking studies are often eagerly awaited. The municipalities are interested in using them to compare and improve their web sites. To enable more targeted use of the results, more detailed information is needed.

To supplement the large scale web accessibility results, the eAccessibility Checker has been developed.<sup>13</sup> The users themselves can use this online tool to evaluate and detect barriers on a single web page by entering a URL. Figure 4 shows an example of results presented by the eAccessibility Checker. They include additional information such as:

- location of potential problem
- explanations why the issue might represent a barrier
- suggested solution and good practice example
- background material (e.g. references to guidelines)
- ranking list of evaluated web sites

This allows web developers to get immediate feedback on their implementation, including how the barriers can be fixed. In the future, the tool could be integrated more tightly in a web development cycle as suggested in [16].

The tool can also be used by web site editors and owners. However, for editors who are not very familiar with (X)HTML and CSS, there exists a challenge with this approach. It is not always easy to understand which problems can be fixed by the web editors and which barriers are located in the templates of the content management systems and therefore need to be fixed by the web developers.










Most existing content management systems (CMS) require the editors to have expert knowledge on accessibility to produce accessible web content, while only few facilitate accessibility [17].

Coming back to the example of alternative text for images, existing CMS handle this quite differently. On the one hand there are systems where it is not at all possible to enter alternative texts for images. On the other hand some systems force the editors to add alternative text whenever images are uploaded. In the third set of CMS editors can choose to add alternative texts to images or not. The editors need to be aware of the web accessibility features of the CMS.

How the barrier can be removed depends on the CMS that is used. There is no universal solution for the problem. We plan to set up a wiki where developers and experts of different content management systems can submit descriptions. The information on how to fix the barriers will be linked to the results presented by the tool. A similar approach is implemented in the Dutch “Web Guidelines Quality Model” [18].

---

<sup>13</sup> The checker is available at <http://accessibility.egovmon.no/>.

Failed applied tests: 7		hide failed tests 
Invalid style-declaration found. (2 occurrences)	expand 	
Found invalid code: end tag for "ul" which is not finished (1 occurrence)	expand 	
Found invalid code: document type does not allow element "ul" here; missing one of "object", "applet", "map", "iframe", "button", "ins", "del" start-tag (1 occurrence)	expand 	
Found form control element without id. (3 occurrences)	expand 	
Passed applied tests: 119		hide passed tests 
Global: No <embed> element found.	expand 	
Global: DOCTYPE declaration valid	expand 	
Global: This page uses latest W3C technologies.	expand 	

**Fig. 4.** The eAccessibility Checker present a list of results. Each result is linked to further details.

## 6 Conclusion and future work

The eGovMon tool has a two-fold strategy for presenting accessibility results. It provides both survey results from large scale evaluations and an interface for detecting barriers in single web pages. This strategy makes it possible to both provide data on a high level – e.g. how accessible is my county compared to others, and the possibility to find individual barriers on the evaluated pages.

DIFI / Norge.no provides a yearly benchmarking survey on the quality of public Norwegian web sites, including accessibility. In contrast to eGovMon, these measurements are done manually by trained experts.

Even though the two methodologies both aim at measuring accessibility, there are many differences. Indeed, there are in total only three overlapping tests. Despite of this, we have shown that there is a correlation between the results produced by the two different methodologies. Web sites which receive a good or bad result in one of the survey are very likely to get a similar result by the other.

## References

1. European Commission, DG Information Society and Media: Study on user satisfaction and impact in EU27. Accessed May 2009. (2008) [http://ec.europa.eu/information\\_society/activities/egovernment/studies/docs/user\\_satisfaction\\_final\\_report.pdf](http://ec.europa.eu/information_society/activities/egovernment/studies/docs/user_satisfaction_final_report.pdf).

2. Agency for Public Management and eGovernment (DIFI): Quality of public web sites. Accessed February 2009. (2008) <http://www.norge.no/kvalitet/>.
3. Web Accessibility Benchmarking Cluster: Unified Web Evaluation Methodology (UWEM 1.2). Accessed February 2009. (2007) [http://www.wabcluster.org/uwem1\\_2/](http://www.wabcluster.org/uwem1_2/).
4. Snaprud, M., Sawicka, A.: Large Scale Web Accessibility Evaluation - A European Perspective. In Stephanidis, C., ed.: HCI (7). Volume 4556 of Lecture Notes in Computer Science., Springer (2007) 150–159
5. World Wide Web Consortium: Web Content Accessibility Guidelines 1.0. W3C Recommendation 5 May 1999. <http://www.w3.org/TR/WCAG10/> (1999)
6. Nietzio, A., Strobbe, C., Velleman, E.: The Unified Web Evaluation Methodology (UWEM) 1.2 for WCAG 1.0. [19] 394–401
7. Nálevka, P., Kosek, J.: Relaxed – on the way towards true validation of compound documents. In: Proceedings of WWW. (2006)
8. Casado, C., Martinez, L., Olsen, M.G.: Is it possible to predict the manual web accessibility results using the automatic results? In: Human Computer Interaction 2009 (to appear). (July 2009)
9. Ulltveit-Moe, N., Olsen, M.G., Pillai, A., Thomsen, C., Gjøsæter, T., Snaprud, M.: Architecture for large-scale automatic web accessibility evaluation based on the UWEM methodology. In: Norwegian Conference for Informatics (NIK). (November 2008)
10. World Wide Web Consortium: Evaluation and Report Language (EARL) 1.0 W3C Working Draft 23 March 2007. <http://www.w3.org/TR/EARL10/> (2007)
11. Thomsen, C., Pedersen, T.B.: Building a web warehouse for accessibility data. In: DOLAP '06: Proceedings of the 9th ACM international workshop on Data warehousing and OLAP, New York, NY, USA, ACM (2006) 43–50
12. Brajnik, G., Lomuscio, R.: SAMBA: a semi-automatic method for measuring barriers of accessibility. In Pontelli, E., Trewin, S., eds.: ASSETS, ACM (2007) 43–50
13. Cullen, K., Kubitschke, L., Meyer, I.: Assessment of the status of eAccessibility in Europe. Accessed February 2009. (2007) [http://ec.europa.eu/information\\_society/activities/einclusion/library/studies/meac\\_study/index\\_en.htm](http://ec.europa.eu/information_society/activities/einclusion/library/studies/meac_study/index_en.htm).
14. Nomensa: United Nations global audit of web accessibility. <http://www.un.org/esa/socdev/enable/documents/fnomensarep.pdf> (2006)
15. Bühler, C., Heck, H., Nietzio, A., Olsen, M.G., Snaprud, M.: Monitoring Accessibility of Governmental Web Sites in Europe. [19] 410–417
16. Brajnik, G.: Comparing accessibility evaluation tools: a method for tool effectiveness. *Univers. Access Inf. Soc.* **3**(3) (2004) 252–263
17. Nedbal, D., Petz, G.: A Software Solution for Accessible E-Government Portals. [19] 338–345
18. Overheid heeft Antwoord: The web guidelines quality model. Accessed February 2009. (2009) <http://www.webrichtlijnen.nl/english/>.
19. Miesenberger, K., Klaus, J., Zagler, W.L., Karshmer, A.I., eds.: Computers Helping People with Special Needs, 11th International Conference, ICCHP 2008, Linz, Austria, July 9–11, 2008. Proceedings. In Miesenberger, K., Klaus, J., Zagler, W.L., Karshmer, A.I., eds.: ICCHP. Volume 5105 of Lecture Notes in Computer Science., Springer (2008)



# A Rule-based Approach for Semantic Consistency Management in Web Information Systems Development<sup>\*</sup>

Francisco J. Lucas, Fernando Molina, and Ambrosio Toval

Software Engineering Research Group  
Department of Informatics and Systems  
University of Murcia (Spain)  
fjlucas@um.es, fmolina@um.es, atoval@um.es

**Abstract.** Consistency problems are mainly due to the existence of multiple views (models) for the same system, which may contain a contradictory joint description. A scope in which these consistency problems can also appear is that of Web Engineering, which focuses on the application of sound practices for the development of web-based information systems (WIS). The aim of this paper is to show how a rigorous approach based on the concepts of models transformations and rewriting logic can offer a suitable framework to manage (analysis and handling) the semantic model consistency in this scope. To illustrate this approach, an application example is shown, which analyzes different consistency problems that can appear between WIS navigational models and usability models.

## 1 Introduction

Consistency problems have existed in Information System (IS) development since its beginning. This is mainly due to the existence of multiple views (models) for the same system, which may potentially contain contradictory specifications of that system. These inconsistencies among different models or views of a system may be a source of numerous errors in the software system [1] and, moreover, may consequently complicate the software management [2].

Furthermore, in recent years, these inconsistency problems have become more important as a result of the profound impact of the Model Driven Engineering (MDE) approach [3] and, particularly, the Model Driven Architecture (MDA) proposal [4], in which the use of models guides the development of a system. Within this scope, these problems

---

<sup>\*</sup> Partially financed by the Spanish Ministry of Science and Technology, project DEDALO TIN2006-15175-C05-03 and MELISA-GREIS (PAC08-0142-335). Fernando Molina is partially funded by the Fundación Séneca (Región de Murcia).

could make the use of models as a source of automatic code generation impossible [5].

A scope in which these consistency problems can also appear is that of Web Engineering [6], which focuses on the application of sound practices for the development of web-based information systems (WIS). In recent years, numerous methodologies for WIS development have appeared and, most of them, are aligned with the model driven engineering approaches. Although each methodology has special features that make it different from the rest, in general, each one proposes the use of different models to design the different views of a WIS like content, navigation or functionality. These models drive the development process and serve as a basis for the construction of the WIS. The existence of different views for the same WIS can cause the appearance of consistency problems among the different models of the system or even within each model.

The aim of this paper is to show how a rigorous approach based on the concepts of models transformations and rewriting logic [7] can offer a suitable framework to manage (analysis and handling) the semantic model consistency in this scope automatically, only defining transformation rules. Maude [8] is the language used to specify the presented framework, where transformation rules are used to define the models behaviour and the inconsistency handling, and commands and inference mechanisms of Maude are used to simulate the behaviour and to define the inconsistency checking and handling. The concepts of models transformations are used for linking the approach to the industrial development. To illustrate this approach, an application example is shown, which analyzes different consistency problems that can be checked and fixed between WIS navigational models and usability web models.

The remainder of the paper is structured as follows. In Section 2, the main definitions used in the paper as well as the main features of WIS development and Maude are shown. Section 3 explains the approach for managing inconsistency problems by means of transformation languages and Section 4 shows through an application example how the approach can be used. Finally, related work, conclusions and further work are presented.

## **2 Background**

### **2.1 Model Consistency Concepts**

A review of the existing bibliography about consistency reveals several possible definitions on model consistency, due to the fact that these concepts are used in different, even in ambiguous or contradictory ways in

different contexts [9,10]. With the aim of unifying the terminology used in the remainder of the paper, the definitions and sources adopted for each concept related to consistency are as follows:

- Consistency. A state in which two or more elements, which can be overlapped in different models of the same system, have a satisfactory description regarding to a set of defined consistency rules [11].
- Syntactic consistency. This kind of consistency should guarantee that a model conforms to its abstract syntax (specified by its metamodel) [12].
- Semantic consistency. This consistency requires that models behaviour be semantically compatible [12]. The approach presented in this paper is focused on this kind of inconsistency problems.

## **2.2 Web Information Systems Development**

In the scope of Web Engineering, numerous methodologies have arisen with the aim of helping in the development of WIS that satisfies the quality requirements demanded by their users. Some examples of these methodologies are WebML [13], OO-H [14] or UWE [15], to name a few. As it was aforementioned, generally all of them have proposed to carry out the WIS development through the use of a set of models, being each one of them focused on one of the views of the WIS. The models that are used by almost any methodologies are navigational models (to model the interaction between users and the WIS), behaviour models (to model the functionality offered by the system) and presentation models (to represent features related to the final presentation of the system).

In this kind of systems, some quality attributes as usability become critical [16], which has motivated the appearance of quality models that are used together with navigational or presentation models in order to evaluate the quality of the designed WIS at modelling time. Thus, these quality models add a new possible source of inconsistencies because they establish some constraints that can be violated for other models. As it will be shown in Section 4, these constraints can be related to the distances between related nodes, levels of importance of the information presented in the system, etc. and they will be used to illustrate our approach for inconsistency management in WIS.

## **2.3 The Formal Language Maude**

The formal language chosen to develop our approach is Maude [8]. This language is based on equational and rewriting logic. In rewriting logic, a

system is specified through a rewrite theory, which consists of a signature  $\Sigma$  (sorts and operations), a set  $E$  of equations, and a set of rewriting rules. The static part of a system is modelled by means of equational logic ( $\Sigma$  and  $E$ ), and the dynamic part is specified by adding rewriting rules, that is, a set of rules that specify how the state of the system changes.

One of the most important concepts of rewriting logic that will be used in our approach is that of the rewriting rule. A rewriting rule (named  $l$ ) describes a local concurrent transition that can take place in a system. If the pattern in the left-hand side of the rule ( $t$ ) matches a fragment of the system state, the matched fragment is transformed into the corresponding state of the right hand side of the rule ( $t'$ ), which is expressed as:  $l : t \rightarrow t'$ . In the context of this paper, this concept can be used directly to implement transformation rules.

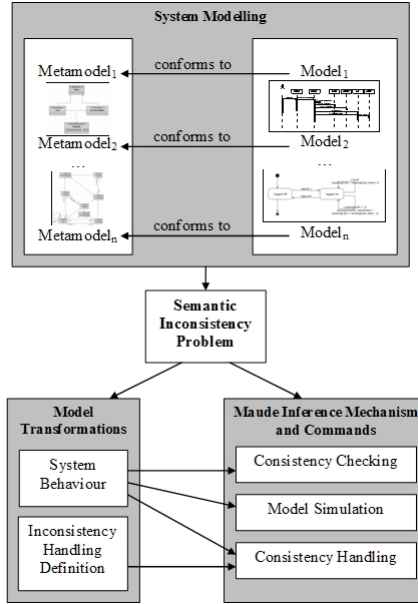
Maude allows both the specification of this kind of logic and its execution in an object oriented manner, in which the system elements are represented as classes that can be instantiated as objects. In addition, Maude also offers several extension modules and features that can be used in the context of consistency management.

### 3 Semantic Consistency Management based on Model Transformation Rules

#### 3.1 Approach Overview

The semantic model consistency is related to the correct behaviour that the system must have once it has been implemented. Figure 1 summarizes the main elements of which the presented approach is made up. The main aim of this paper is to show how rewriting rules can be used as model transformations rules in order to manage inconsistency problems in WIS development. The definition of a transformation is usually expressed as a set of transformation rules that describe how a source model is transformed into a target model [17]. That is, these rules describe how meta-model elements of the source model are transformed into corresponding metamodel elements in the target model. As we will see in next sections, in our approach the rewriting rules of Maude will specify the necessary model transformations to define and handle inconsistency problems in WIS.

From our point of view, since the semantic inconsistency problems are related to the behaviour defined by the semantics of the different models (metamodels) involved in the development, transformation rules will be



**Fig. 1.** Approach overview

used firstly for expressing how the system behaviour, represented by models, changes. In general, this definition may involve any number of meta-models. Note that, since different semantic inconsistency problems can be defined over the same behaviour, the same behaviour definition can be used for managing all them. At this point, Maude’s formal tools/facilities for analysis are used to execute the transformation rules and check all these problems. Maude also allows us to use this behaviour definition to execute system prototypes.

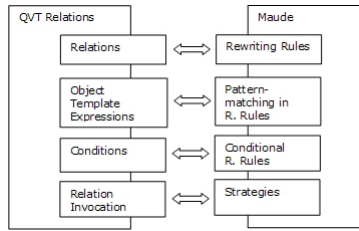
Once an inconsistency has been found, in our approach model transformations are used again to define how it will be handled. Maude is used again to execute the rules and to handle the inconsistency.

Reinterpreting an inconsistency problem this way, the use of Maude and the concepts of model transformations seems a suitable way for the definition of consistency relationships among models. This process will be illustrated by means of an example in Section 4.

### 3.2 Model Transformations Features in Maude

Figure 1 shows the main elements that make up the approach, and how model transformations will be used to define the system behaviour and the

inconsistency handling. Several transformation languages have appeared in the scope of MDE [18,19,20]. In the scope of MDA, *QVT Relations* [21] is one of the languages defined by the OMG and is the most abstract and user-friendly language of the languages defined within the QVT standard. For these reasons, we will use its features to show how they can be expressed by means of the strengths offered by Maude (see Figure 2).



**Fig. 2.** Model Transformation Formalization

QVT Relations is a declarative models transformation language, so its implementation over a declarative language like Maude is more ‘natural’ than with other non-declarative languages.

On the one hand, a transformation is expressed in QVT Relations by means of relations between metamodel elements. A relation declares constraints that must be satisfied by the two or more metamodels (or domains) that participate in the relation, that is, it specifies a relationship that must hold between the elements of the candidate models. Each domain establishes a pattern (with a set of variables and constraints) that must be matched with the candidate models in order to execute the transformation, known as *object template expressions*. These templates also serve for creating new objects in the target model. Furthermore, a relation can also have two set of predicates: a *when* clause and a *where* clause, which are used to define pre-conditions and post-conditions of the relation, respectively.

Figure 2 summarizes how the specification of QVT transformations can be carried out using Maude. The transformation rules, named relations in QVT, can be specified as Maude rewriting rules that change and create the elements of the target model. The object patterns defined in QVT are directly expressed in Maude, since this language offers pattern-matching in the terms simplification.

Finally, in order to specify the constraints, we differentiate two kinds of them: those that express conditions over the models, which will be

specified as conditions in the rewriting rules; and those that *guide* the execution of a transformation (top-level and non-top-level relations and invocations from *when* or *where* clauses) that will be specified through the Maude Strategy Language [22].

Note that QVT Relations provides all these elements in a textual way, but once they are specified in Maude, they have been transformed into mathematical entities and we can take advantage of all the power of mathematical inference mechanisms, without losing the intuition of the QVT concepts.

All these features will be explained in depth in the next section.

## 4 Semantic Consistency Management in WIS

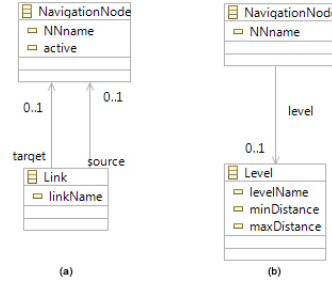
This section shows how the presented approach can be used for managing inconsistencies in WIS scope. To do this, an example of a consistency relationship among navigational models and usability models is defined. There are numerous possible semantic inconsistency problems between these kinds of diagrams. In this example, we want to check that the usability constraints expressed in a usability model are hold by the navigational model nodes and the traces for reaching them. That is, if an importance level is established for a node, the navigation in the system must allow reaching that node in the number of steps that its level indicates. All these concepts are explained in depth in next subsections.

### 4.1 Involved Metamodels

Before defining how the inconsistency problem will be checked and handled, it is necessary to show the metamodels involved. In this example, two metamodels are involved but, in general, more than two metamodels can appear, or the inconsistency problem may even involve only one metamodel.

**4.1.1 Navigational Metamodel** In this work the first metamodel that we will use is a simple navigational diagram metamodel. This metamodel will be specified in Maude, and its models will be instantiated through *Maude objects*. Figure 3 (a) shows this metamodel.

Two elements appear in this metamodel. One key element is the meta-class `NavigationNode` that models a piece of information, represented by a web page. The second element is the metaclass `Link` that represents directed relations, which indicate the possible routes that could be followed



**Fig. 3.** (a) Navigational metamodel (b) Usability metamodel

from here on. Other elements can appear in navigational metamodels such as menus, indexes, etc. but they have not been included to facilitate readers the understanding of our proposal.

Finally, another element has been added to this metamodel that does not appear in the “standard” navigational metamodel, but that it is necessary to define the behaviour through a transformation rule. This element is the attribute **active** of a node that marks the current node of a model when its behaviour is being simulated.

**4.1.2 Usability Metamodel** Regarding to the usability features mentioned in previous sections, we use a simplified usability features metamodel to express them (see Figure 3 (b)). In this metamodel, **NavigationNode** references the nodes appearing in the navigational models. This element will be used as a link point between the two metamodels.

Other important element of this metamodel is the metaclass **Level**. This element and its attributes are used to represent the concept of importance of a node and its link with the **NavigationNode** entity allows modellers to label each node with an importance level. Each **Level** has three attributes: a name and the integers **maxDistance** and **minDistance**, which define the minimum and maximum distance between the nodes labelled with a **Level** and the node that represents the entry point to the WIS. This entity is often used in usability models to express that all the information presented in the WIS has not the same importance, which introduce some constraints in navigational models that must be assured. Interested readers can find a detailed rationale behind these usability features in [23].



## 4.2 Behaviour Definition

Once the metamodels have been shown, the behaviour that will be used to check semantic inconsistency problems has to be defined. This behaviour is based on the simulation of the navigational model using, on the one hand, the active node and, on the other hand, one of its outgoing links. Since the execution rules are non-deterministic, different final states can be reached from a node. Note that since the semantic inconsistency problems are related to the specific behaviour defined, different problems can be managed based on the same behaviour definition. Figure 4 shows the rule that simulate a navigational model. To do this, the attribute `active` of node is used. This rule makes use of a *Message* (a sort of *Maude*), named *simulateBehaviour*, that will be used to indicate to Maude the operation that we want to carry out.

```

*** Behaviour definition.
op simulateBehaviour : -> Msg [ctor] .

rl [NavigationBehaviour] :
  simulateBehaviour
  < l0id : Link | linkName : ln, source : nn0id1, target : nn0id2 >
  < nn0id1 : NavigationNode | active : true, s1:AttributeSet >
  < nn0id2 : NavigationNode | active : false, s2:AttributeSet >
=> simulateBehaviour
  < l0id : Link | linkName : ln, source : nn0id1, target : nn0id2 >
  < nn0id1 : NavigationNode | active : false, s1:AttributeSet >
  *** The new active node becomes the target node of the link.
  < nn0id2 : NavigationNode | active : true, s2:AttributeSet > .

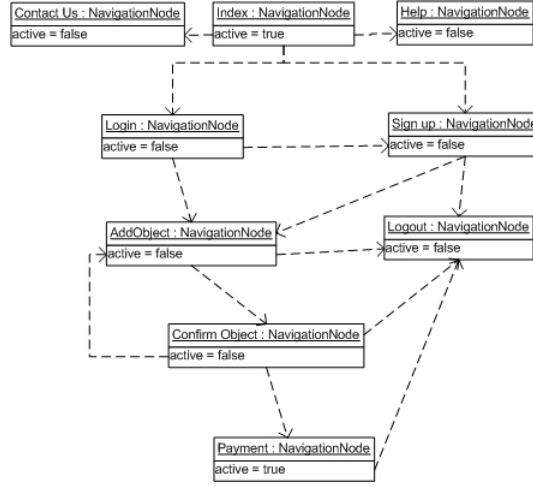
```

Fig. 4. Behaviour definition

## 4.3 Consistency Checking

This section shows how to check the consistency problem previously defined by means of the *search* command of Maude. This command allows us to use the previous rule to make reachability checks in a navigational model. Since the level of a node establishes the maximum distance that can exist between this node and the initial one, we will use this command to check if we can reach that node, and how many steps we have to do in order to reach it. If this number of steps is greater than the maximum distance defined for the node, an inconsistency problem exists.

The application example chosen to illustrate our approach deals with an interaction for a simple on-line shop cart system. Figure 5 shows a



**Fig. 5.** Example of navigational model for an online shopping system

navigational diagram that models how to register and buy in this system. In this model the initial node is named 'Index'. Moreover, Figure 6 shows the importance levels that the modeller has defined and how they are linked to each node. In order to improve the usability of the WIS, those nodes considered important have been labelled with a higher importance level, which implies that they must be near to the entry point of the WIS in order to ease the users' navigation in the system.

Once the models have been defined, the rule shown in Figure 4 is used together with the Maude search command to check the existence of inconsistency nodes, that is, nodes that violate the constraints defined in the usability model of 6. Figure 7 (a) shows how the search command is used to check the node *Payment*. This command verifies the reachability of a node named *Payment* using the rule previously defined. If this node is reachable in some way using that rule, its attribute *active* will have the value *true*. Finally the Figure 7 (b) shows a fragment of the output produced by the command, which shows what a solution exists, that is, the node is reachable and the number of states (steps), four, to reach it. This way the *Payment* node is inconsistent regarding to its level whose maximum distance to the 'Index' node is three. This information together with the rules defined in the next section will be used to offer users the possibility of handling this inconsistency.

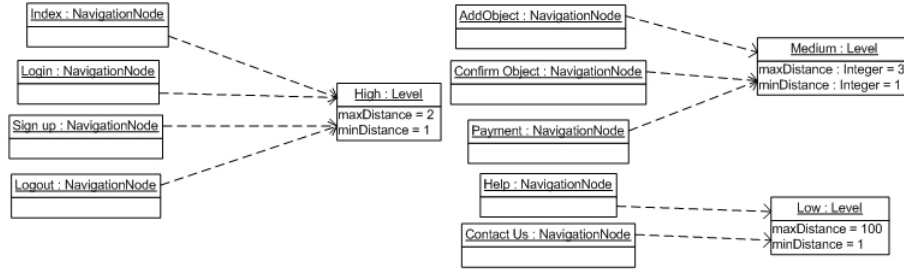


Fig. 6. Example of usability model

```
(a)
search [1] simulateBehaviour diagram =>* C:Configuration
  < nn0id:0id : NavigationNode | NNname : "Payment" , active : true >
  < nn0id:0id : NavigationNode | NNname : "Payment" , level : lvl:0id >
  < lvl:0id : Level | levelName : "Medium", s:AttributeSet > .

(b)
Solution 1 (state 4)
states: 5  rewrites: 9 in 464618638ms cpu (26ms real) (0 rewrites/second)
C:Configuration --> simulateBehaviour
< Low : Level | levelName : "Low", minDistance : 1, maxDistance : 100 >
< High : Level | levelName : "High", minDistance : 1, maxDistance : 2 >
< Index : NavigationNode | NNname : "index", active : false > ...
< l1 : Link | linkName : "3->4", source : ConfirmObject, target : Payment >
nn0id:0id --> Payment
lvl:0id --> Medium
s:AttributeSet --> minDistance : 1, maxDistance : 3
```

Fig. 7. (a) Search command, (b) Search command output

#### 4.4 Consistency Handling

To specify the inconsistency handling, we assume that the problem has been already checked. In this example, once the inconsistency has been found, several possible solutions appear. To illustrate our approach, the handling rules proposed are based on adding new links that connect the inconsistent node with another node or with the index node of the model. These rules assume that the problem has been found and use the attribute **active** of **NavigationNode**, to match the inconsistent elements. Figure 8 (a) shows the rule that links the inconsistent node with another. To do this, the user makes use of a new message, *addLink*, which allow us to indicate the source and target nodes of the link. This way the user can create links among nodes and handle the inconsistency. Figure 8 (b) shows

the rule that, in the same way as the previous one, links the *Index* node with the inconsistent node.

```
(a)
rl [InconsistencyHandling1] :
  addLink(source:String, target:String)
  < nn0id1 : NavigationNode | NNname : source:String, s1:AttributeSet >
  < nn0id2 : NavigationNode | NNname : target:String, s2:AttributeSet >
=>
  < nn0id1 : NavigationNode | NNname : source:String, s1:AttributeSet >
  < nn0id2 : NavigationNode | NNname : target:String, s2:AttributeSet >
  < oid(source:String + target:String) : Link |
    linkName : source:String + "->" + target:String,
    source : nn0id1, target : nn0id2 > .

(b)
rl [InconsistencyHandling2] :
  addLinkIndex2(node:String)
  < nn0id1 : NavigationNode | NNname : "index", s1:AttributeSet >
  < nn0id2 : NavigationNode | NNname : node:String, s2:AttributeSet >
=>
  < nn0id1 : NavigationNode | NNname : "index", s1:AttributeSet >
  < nn0id2 : NavigationNode | NNname : node:String, s2:AttributeSet >
  < oid(node:String + "index") : Link |
    linkName : "index" + "->" + node:String,
    source : nn0id1, target : nn0id2 > .
```

**Fig. 8.** Rules for handling an inconsistency (a) Adding a new link (b) Adding a new link from the *Index* node.

These two rules can be used by users to handle this kind of inconsistency. Figure 9 shows a fragment of the Maude output when the inconsistency is handled. To do that, the message *addLink* and the rewrite command of Maude are used to add a link between the *Login* and *Payment* nodes in the model.

## 5 Related Work

The inconsistency problems have been tackled in a great deal of work (see [10]). In the scope of UML development, the most frequently tackled problems are related to syntactic features and only a few of them handle semantic consistency problems.

With regard to the formal analysis of consistency in WIS scope, to our knowledge extent, there does not exist specific work that tackle it,

```

Maude> rew diagram addLink("Login", "Payment") .
rewrite in EX1 : diagram
addLink("Login", "Payment") .
rewrites: 1 in 1628036047000ms cpu (0ms real) (0 rewrites/second)
result Configuration: addLink("Login", "Payment")
< Login : NavigationNode | NNname : "login", active : false >
< Payment : NavigationNode | NNname : "Payment", active : false >
...
< oid("loginPayment") : Link | linkName : "login->Payment",
  source : Login, target : Payment >

```

**Fig. 9.** (a) Fragment of the Maude output when the inconsistency is handled

although there exists some work dedicated to the application of formal methods in WIS verification and validation. The authors of [24] use Haskell to offer automatic verification over Web sites regarding to the correctness of a set of rules. Differently from our work, [24] is focused on the code of web pages, whereas the presented approach is focused on finding inconsistency problems in early phases of software development such as modelling. Moreover, [24] does not tackle system behaviour problems related to the usability features of the system.

[25] also uses rewriting techniques to model dynamic behaviour of Web sites. However, the verification checks shown in it can not be used at modelling time. Verifications on a WIS already implemented imply more cost when errors have to be fixed. Moreover, our work deals with usability features as well as dynamic behaviour problems.

The authors of [26] tackle usability problems during the navigational modelling. To do that, StateWebCharts notation is used. This kind of notation is not well-known in the WIS community and, for this reason, we think that our approach is more usable due to the use of the *de facto* WIS standard diagrams to model the navigation in a WIS.

Finally, [27] also uses Maude to verify properties over WIS using its model-checking tool. The properties are related to secure access and connectivity. In the same way that other previously commented approaches, the behaviour checking is carried out on a set of web pages, instead of making similar checks at modelling time.

## 6 Conclusions

The research presented shows the feasibility of using formal techniques (algebraic specifications and Maude) in order to manage semantic inconsis-

tency problems in Web Engineering. The metamodel specifications made in this approach offer a powerful way to verify type properties and the correctness of the models without losing the legibility, intuition, and expressivity of others transformation languages. Furthermore, in the formal framework proposed the inconsistency problems in WIS development are expressed by means of mathematical entities, so we can take advantage of all the power of mathematical inference mechanisms of Maude. This allows us to apply commands and extensions to infer information about those problems and use it for handling them.

With regard to future work, on the one hand, this approach is being integrated within the MOMENT2 framework [28] in order to improve its CASE support. Moreover, the features of MOMENT2 are also being extended to improve the support to this kind of problems, for example, since there is no way of parameterizing the rules in MOMENT2 and the execution of the rules is non-deterministic, as in Maude, we can not indicate which inconsistent elements of a model have to be handled. In this paper this is made by means of a message introduced in the definition of the handling rule, but in MOMENT2 only elements of metamodels/models can appear in the rules. For this reason, we are working together with the authors' MOMENT2 to extend its functionality. On the other hand, the techniques presented in this paper are being applied to technologies used in real web developments. Specifically the inconsistency problems managed in this paper are being used in some industrial case studies to verify some similar problems over the navigation rules of JSF [29].

## References

1. Muskens, J., Bril, R., Chaudron, M.: Generalizing consistency checking between software views. 5th Working IEEE/IFIP C. on Software Architecture (2005)
2. Huzar, Z., Kuzniarz, L., Reggio, G., Sourrouille, J.L., eds.: Proc. of Workshop on consistency Problems in UML-based Software Development II. (2003)
3. Schmidt, D.: Guest editor's introduction: Model-driven engineering. IEEE Computer **39** (2006) 25–31
4. OMG: MDA Guide Version 1.0.1, <http://www.omg.org/mda>. (2001)
5. Simmonds, J., M. Bastarrica, C.: A tool for automatic UML model consistency checking. 20th IEEE/ACM I. C. on Automated software engineering. USA (2005)
6. Ginige, A., Murugesan, S.: Guest editors' introduction: Web engineering - an introduction. IEEE MultiMedia **8** (2001) 14–18
7. Ehrig, H., Mahr, B.: Fundamentals of Algebraic Specification. Equations and Initial Semantic. ISBN 3-540-13718-1 Springer-Verlag. 1985 (1985)
8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcote, C.: Maude 2.4 Manual., <http://maude.csl.sri.com/>. (2008)
9. Shinkawa, Y.: Inter-model consistency in uml based on cpn formalism. XIII Asia Pacific Software Engineering Conference (APSEC'06), 2006 (2006)

10. Lucas, F.J., Molina, F., Toval, A.: A Systematic Review of UML Model Consistency Management. IST Special Issue on Quality of UML Models (To appear)
11. Spanoudakis, G., Zisman, A.: Inconsistency management in software engineering: Survey and open research issues. In Chang S. K., editor, *Handbook of Software Engineering and Knowledge Engineering*. (2001)
12. Engels, G., Küster, J.M., Heckel, R., Groenewegen, L.: A methodology for specifying and analyzing consistency of object-oriented behavioral models. In *Proceedings of eighth ESEC held jointly with FSE2001* ACM Press, Vienna, Austria (2001)
13. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: *Designing Data-Intensive Web Applications*. Morgan Kaufmann Inc., San Francisco, CA, USA (2002)
14. Gómez, J., Cachero, C., Pastor, O.: Conceptual modeling of device-independent web applications: Towards a web engineering approach. *IEEE Multimedia* **8** (2001) 26–39
15. Koch, N., Kraus, A.: The expressive power of uml-based web engineering. 2nd Int. Workshop on Web-oriented Software Technology (IWWOST) (2002)
16. Juristo, N., Moreno, A.M., Sánchez, M.I.: Analysing the impact of usability on software design. *Journal of Systems and Software* **80** (2007) 1506–1516
17. Kleppe, A.G., Warmer, J., Bast, W.: *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley (2003)
18. SmartQVT: SmartQVT. <http://smartqvt.elibel.tm.fr/> (2007)
19. ikv++ technologies ag: Medini QVT 1.1 . <http://www.ikv.de/> (2007)
20. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: Atl: a qvt-like transformation language. 21th ACM SIGPLAN, OOPSLA, Oregon, USA (2006)
21. OMG: MOF QVT Final Adopted Specification. Object Management Group., Retrieved from: <http://www.omg.org/docs/ptc/07-07-07.pdf>. (2007)
22. Eker, S., Martí-Oliet, N., Meseguer, J., Verdejo, A.: Deduction, strategies, and rewriting. *Electr. Notes Theor. Comput. Sci.* **174** (2007) 3–25
23. Molina, F., Álvarez, A.T.: A generic approach to improve navigational model usability based upon requirements and metrics. In: *WISE Workshops*. (2007)
24. Ballis, D., García-Vivó, J.: A rule-based system for web site verification. *Electr. Notes Theor. Comput. Sci.* **157** (2006) 11–17
25. Lucas, S.: Rewriting-based navigation of web sites: Looking for models and logics. *Electr. Notes Theor. Comput. Sci.* **157** (2006) 79–85
26. Winckler, M., Barboni, E., Palanque, P.A., Farenc, C.: What kind of verification of formal navigation modelling for reliable and usable web applications? *Electr. Notes Theor. Comput. Sci.* **157** (2006) 207–211
27. Flores, S., Lucas, S., Villanueva, A.: Formal verification of websites. *Electr. Notes Theor. Comput. Sci.* **200** (2008) 103–118
28. Boronat, A.: Moment2. <http://www.cs.le.ac.uk/people/aboronat/tools/moment2> (2008)
29. Microsystems, S.: Javaser faces. <http://java.sun.com/javaee/jaserverfaces/> (2009)





# Slicing microformats for information retrieval<sup>\*</sup>

J. Guadalupe Ramos<sup>1</sup>, Josep Silva<sup>2</sup>, Gustavo Arroyo<sup>2</sup>, and Juan C. Solorio<sup>1</sup>

<sup>2</sup> DSIC, Universidad Politécnica de Valencia  
Camino de Vera s/n, E-46022 Valencia, Spain.

{jsilva,garroyo}@dsic.upv.es

<sup>1</sup> Instituto Tecnológico de La Piedad  
Av. Tecnológico 2000, La Piedad, Mich., México. CP 59300  
{guadalupe@dsic.upv.es, juancsol@hotmail.com}

**Abstract.** *Microformats* are a medium to incorporate semantic information into the web by means of standard tags which are enriched with particular attributes. They are a set of simple and open data formats built upon existing and widely adopted standards, hence, they are considered a pragmatic path to the Semantic Web.

In this work, we introduce a new method for information extraction from the semantic web. Basically we model the semantic information, which is contained in a set of web pages, in a formal graph like structure, namely, semantic network. Then, we introduce a novel slicing based technique for information extraction from semantic networks. In particular, the technique allows us to extract a portion—a *slice*—of the semantic network with respect to some criterion of interest. The slice obtained represents relevant information retrieved from the semantic network and thus from the semantic web. Our approach can be used to design novel tools for information retrieval and presentation, and for information filtering that was distributed along the semantic web.

## 1 Introduction

The Semantic Web is considered an evolving extension of the World Wide Web in which the semantics of information and services on the web is made explicit by adding metadata. Metadata provides the web contents with descriptions, meaning and inter-relations. The Semantic Web is envisioned as a universal medium for data, information, and knowledge exchange.

Two important technologies for developing the Semantic Web are already in use: The *eXtensible Markup Language* (XML) and the *Resource Description Framework* (RDF) among others [1]. Nevertheless, efforts to extend the Web

---

<sup>\*</sup> This work has been partially supported by the Spanish *Ministerio de Ciencia e Innovación* under grant TIN2008-06622-C03-02, by the *Generalitat Valenciana* under grant GVPRE/2008/001, by the *Universidad Politécnica de Valencia* (Programs PAID-05-08 and PAID-06-08) and by the Mexican *Dirección General de Educación Superior Tecnológica*.

with meaning have gained little traction. These initiatives have been bogged down by complexity and over-ambitious goals, or have simply been too much trouble to implement at a large scale (see, e.g., the discussion in [2]).

Recently, a new initiative has emerged that looks for attaching semantic data to web pages by using simple extensions of the standard tags currently used for web formatting in (X)HTML<sup>1</sup>, these extensions are called *microformats* [3, 4]. A microformat is basically an open standard formatting code that specifies a set of attribute descriptors to be used with a set of typical tags.

*Example 1.* Consider the following XHTML code that introduces information of a common personal card.

```
<h2>Directory</h2>
<p> Vicente Ramos <br>
    Software Development <br>
    118, Atmosphere St. <br>
    La Piedad, México <br>
    59300 <br>
    +52 352 52 68499 <br>
</p>
<h4>His Company</h4>
<a href="page2.html">Company Page </a>
```

Now, let us see the same information but taking into account the standard hCard microformat [5], which is useful for representing people, companies, organizations, and places data.

```
<h2>Directory</h2>
<div class="vcard">
  <span class="fn">Vicente Ramos</span>
  <div class="org">Software Development</div>
  <div class="adr">
    <div class="street-address">Atmosphere 118</div>
    <span class="locality">La Piedad, México</span>,
    <span class="postal-code">59300</span>
  </div>
  <div class="tel">+52 352 52 68499</div>
<h4>His Company</h4>
<a class="url" href="page2.html">Company Page </a>
</div>
```

The `class` property qualifies each type of attribute which is defined by the hCard microformat. The code starts with the required main class `vcard` and classifies the information with a set of classes which are auto-explicative: `fn` describes name information, `adr` defines address details and so on.

---

<sup>1</sup> XHTML is a sound selection because it enforces a well-structured format.

Microformats are a clever adaptation of semantic XHTML that makes it easier to publish, index, and extract semi-structured information like tags, calendar entries, contact information, and reviews on the web. Microformats have given rise to the so-called *semantic web*<sup>2</sup> [6]. Indeed, they are considered a pragmatic path towards achieving the vision set forth for the Semantic Web [4].

Both the Semantic Web and the semantic web require new *formal* models, methods and tools to represent and query the embedded information. In the Semantic Web setting the semantic model is based on the notion of *Ontology*. An ontology defines and categorizes classes of concepts and their relations [1].

In contrast, in the semantic web setting, there does not exist a widely accepted model, and thus, the scientific community must do an effort to propose new approaches and formal methods. In this paper we propose the use of *semantic networks* which is a convenient simple model for representing semantic data; and we define a slicing technique for this formalism in order to analyze and filter the semantic web. A semantic network is often used as a form of knowledge representation; and it is formalized as a graph whose vertices represent concepts, and whose edges represent semantic relations between the concepts [7].

Once the information is modeled in a semantic network, formal methods for information extraction are needed to ensure a systematic and sound treatment of the information. Because semantic networks are implemented as a data structure that contains a considerable amount of information, its treatment is not a trivial task. We use a slicing technique to reduce the complexity of such a data structure.

*Program slicing* is basically a decomposition technique for the extraction of those program statements—the *slice*—that (potentially) affect the values computed at some point of interest. Program slicing was originally introduced by Weiser [8] and has now many applications such as debugging, program specialization [9], and XML filtering [10], see [11] for a survey.

Slicing techniques are (usually) based on a data structure called *Program-Dependence Graph* (PDG) [12]. The PDG allows slicers to find out which sentences of a program are related to some criterion (the so called *slicing criterion*) and thus they belong to the slice.

Therefore, program slicing could be a very convenient way to retrieve information from semantic networks with respect to some slicing criterion. Based on this idea, we introduce a program slicing inspired technique for information extraction from the semantic web. Our technique is based on an extension of semantic network, the *indexed* semantic network, that we conveniently formalize. This new notion of semantic network contains indexes that allow us to extract sub-graphs which are related to a specific topic. Roughly, the technique proceeds as follows: Firstly, an indexed semantic network is built from a collection of web pages. Then, we extract from the indexed semantic network the sub-net which is related to the slicing criterion. Finally, a slice is extracted from the semantic sub-net. The slices extracted from the sub-net represent the semantic informa-

---

<sup>2</sup> Note the different use along the paper of Semantic Web (in capital letters) and semantic web (in lowercase letters).

tion associated to the slicing criterion which, in turn, is the required information by the user.

The main contributions of this paper can be summarized as follows:

- We propose the use of semantic networks to represent semantic webs through the use of microformats, and show its usefulness.
- We extend standard semantic networks with indexes. This extension acts as an interface for the semantic network.
- We introduce a formal slicing based method for information recovering in semantic networks.

The rest of the paper is organized as follows. In Section 2, we overview the topic of semantic networks and recall the basic concepts related to them. In Section 3, we describe how semantic networks can be built from the semantic web. Furthermore, our slicing method for information extraction is formally introduced in Section 4. Finally, in Section 5 we review some related work and conclude.

## 2 Semantic Networks

The concept of *semantic network* is fairly old—in fact, the term of semantic network dates back to Ross Quillian’s works [13] where he introduced it as a way of talking about the organization of human semantic memory—in the literature of cognitive science and artificial intelligence. Nevertheless, it is a common structure for knowledge representation, which is useful in modern and different problems of artificial intelligence. For instance, in the recent Semantic Network Analysis Workshops [14, 15] many applications of this formalism were discussed, e.g., for social networks or hypertext networks.

A semantic network is a directed graph consisting of nodes which represent *concepts* and edges which represent *semantic relations* between the concepts. Sowa [16, 7] introduced a classification of semantic networks, in which the type of *definitional networks* emphasizes the subtype of *is-a* relation between a concept type and a newly defined subtype. This is the kind of semantic network that we will use in this paper. In Figure 1, we present a typical example.

## 3 From the semantic web to the semantic network

Roughly speaking, our method for semantic web information extraction is composed by two main steps:

1. Representing the information in the semantic web with a semantic network
2. Slicing the semantic network

In this section we focus on the first step, while the second one is subject of formal treatment in Section 4.

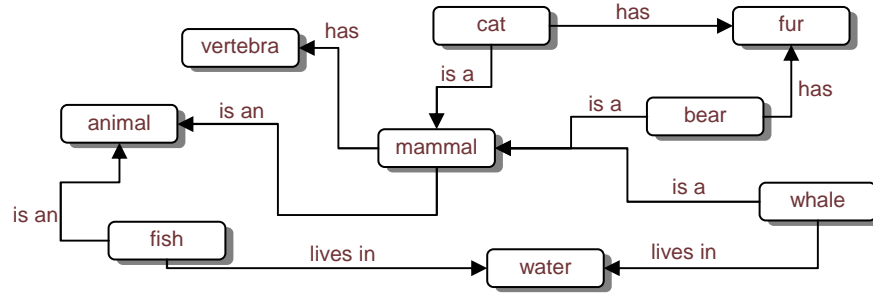


Fig. 1. A definitional semantic network.

### 3.1 Constructing the semantic network from the semantic web

In order to represent semantic information in a semantic network we should decide what is the relevant information to be gathered and what we expect from a web information extraction query. In this work, we consider the microformats, i.e., classes as convenient entities for modeling, and then, for indexing or referencing.

*Example 2.* Let us consider again the semantic microformatted web page code of Example 1. We see that the semantic information is classified by using predefined classes which can embed other classes. For instance the main class `vcard` embeds the `org` class (to define an organization), the `adr` class (to indicate addressing data), etc. The next code shows a semantic web page composed by two main classes, i.e., `vcard` and `vevent` (for events microformatting [17]):

```

<h2>Staff</h2>
<div class="vcard">
  <span class="fn"><strong>Jessica Pechuch</strong></span>
  <p class="role">CEO</p>
  <div class="org">Software Development </div>
  <div class="adr">
    <div class="street-address">Atmosphere 118</div>
    <span class="locality">La Piedad, México</span>,
    <span class="postal-code">59300</span>
  </div>
  <div class="tel">+52 352 52 68499</div>
</div>
<h2>Personal Events</h2>
<div class="vevent">
  <span title="2009-02-25" class="dtstart">
    February 25, 2009
  </span>
  <span class="summary">Microformats use </span> at
  <span class="location"> Main Street 126 </span>
  <div class="description">

```

```

    In this meeting we will discuss the use of microformats
  </div>
</div>

```

In the example we see that microformats use classes to hierarchize the information; thus, classes should be the basic units of our semantic model. If we focus on the relations between classes we identify two kinds of relations, namely:

**strong relations** that are the relations which come from hypertext links between pages or sections of a page by using anchors.

**weak relations** that can be *embedding relationships*, for classes that embeds other classes or *semantic relationships* among classes of the same type, for instance, between two `vcard`.

*Example 3.* Consider again the microformatted code of Examples 1 and 2. From their classes we can build the semantic network depicted in Figure 2 (the grey parts of the figure do not belong to the semantic network and thus they can be ignored for the moment).

In the figure, the nodes of the first page are labeled with  $P1$  and the nodes of the second page are labeled with  $P2$ . Thus, nodes (i.e., concepts) are unique. We observe three kinds of edges: The `locality` class from Example 1 is embedded in the `adr` class. Thus, there is an embedding relationship from node `adr` to node `locality`. Furthermore, `vcard` in  $P1$  and `vcard` in  $P2$  of the semantic web of Example 2 are linked by a semantic relationship. Besides, there is one strong hyperlink to  $P2$  generated by the microformatted tag `<a class="url" href="page2.html">`. Observe that the graph only contains semantic information and their relations; and it omits content or formatting information such as the `<strong></strong>` labels. Observe that we add to the graph two additional concepts,  $P1$  and  $P2$ , which refer to web pages. This is very useful in practice in order to make explicit the embedding relation between microformats and their web page containers.

It is important to note that, in the previous example, similar classes participate in a cyclic relation. This is needed and useful in order to preserve semantic relations among information which is located in many source pages. The source pages to be analyzed in order to build the semantic network should be defined by the user or by the system, for instance, they could be the answer from a web searching engine. Another important design decision is related to the classes to be semantically linked. In the above example we took only main classes, i.e., `vcard` and `vevent`. It was a design decision not to link other classes such as `adr`.

## 4 A technique for information retrieval

In this section we formalize the notions related to semantic networks. Firstly, we define the semantic networks, then we introduce an extension called indexed

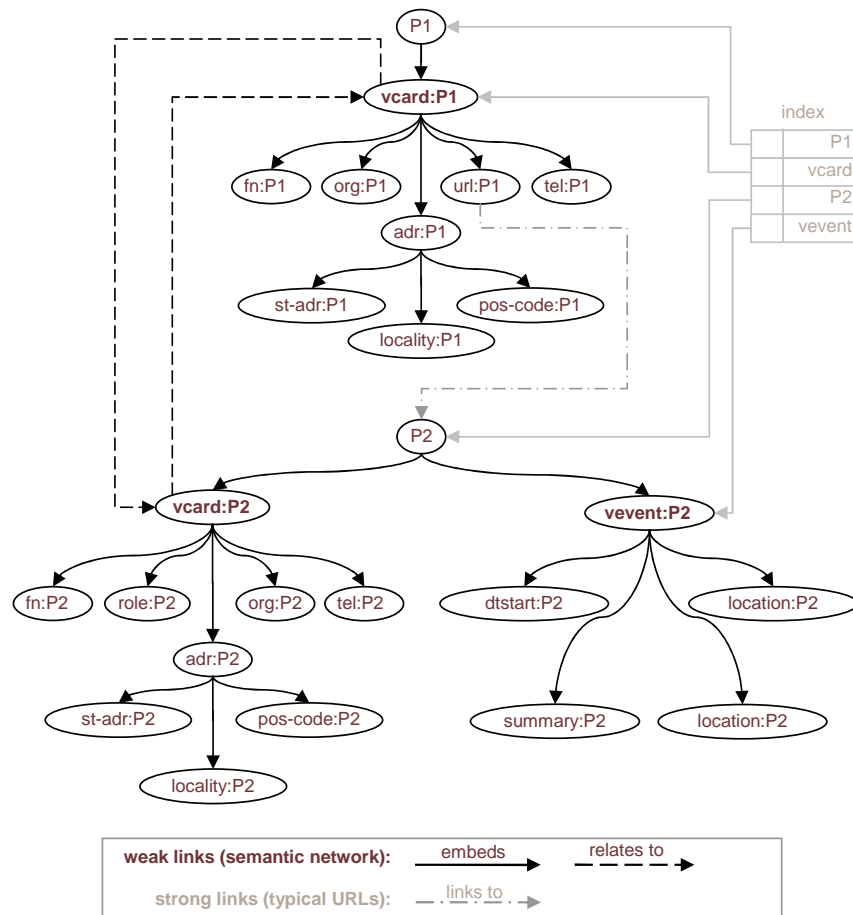


Fig. 2. Semantic network of Example 1 and Example 2.

semantic network. In addition, we define the notion of semantic sub-net. Once, the needed graph structures have been defined, we introduce the concept of backward and forward slicing of such a graphs, and enunciate our fundamental result of semantics preservation of slices. Finally, we show an algorithmic view of our slicing based method for information extraction. Without loss of generality, we only consider weak links (i.e., only semantic relations), thus we analyze semantic networks without taking into account the labels associated to the edges.

#### 4.1 Extending semantic networks

We introduce first some preliminary definitions.

**Definition 1 (semantic network).** *A directed graph is an ordered pair  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  where  $\mathcal{V}$  is a finite set of vertices or nodes, and  $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$  is a set of ordered pairs  $(v \rightarrow v')$  with  $v, v' \in \mathcal{V}$  called edges. A semantic network is a directed graph  $\mathcal{S} = (\mathcal{V}, \mathcal{E})$  in which nodes have been labeled with names of web pages and microformatting classes of these pages.*

As an example of semantic network consider the directed graph in Figure 2 (omitting the grey parts) where nodes are the set of microformatted classes provided by two semantic web pages.

A semantic network is a profuse mesh of information. For this reason, we extend the semantic network with an *index* which acts as an interface between the semantic network and the potential interacting systems. The index contains the subset of concepts that are relevant (or also visible) from outside the semantic net. It is possible to define more than one index for different systems and or applications. Each element of the index contains a key concept and a pointer to its associated node. Artificial concepts such as webpages (See *P1* and *P2* in Figure 2) can also be indexed. This is very useful in practice because it is common to retrieve the embedded (microformatted) classes of each semantic web page.

Let  $\mathcal{K}$  be a set of concepts represented in the semantic network  $\mathcal{S} = (\mathcal{V}, \mathcal{E})$ . Then,  $rnode : (\mathcal{S}, k) \rightarrow \mathcal{V}$  where  $k \in \mathcal{K}$  (for the sake of clarity, in the following we will refer to  $k$  as the *key concept*) is a mapping from concepts to nodes; i.e., given a semantic network  $\mathcal{S}$  and a key concept  $k$ , then  $rnode(\mathcal{S}, k)$  returns the node  $v \in \mathcal{V}$  associated to  $k$ .

**Definition 2 (semantic index).** *Given a semantic network  $\mathcal{S} = (\mathcal{V}, \mathcal{E})$  and an alphabet of concepts  $\mathcal{K}$ , a semantic index  $\mathcal{I}$  for  $\mathcal{S}$  and  $\mathcal{K}$  is any set  $\mathcal{I} = \{(k, p) \mid k \in \mathcal{K} \text{ and } p \text{ is a mapping from } k \text{ to } rnode(\mathcal{S}, k)\}$*

We can now extend semantic networks by properly including a semantic index. We call this kind of semantic network *indexed semantic network* (IS).

**Definition 3 (indexed semantic network).** *An indexed semantic network IS is a triple  $IS = (\mathcal{V}, \mathcal{E}, \mathcal{I})$ , such that  $\mathcal{I}$  is a semantic index for the semantic network  $\mathcal{S} = (\mathcal{V}, \mathcal{E})$ .*



Now, each semantic index allows us to visit the semantic network from a well defined collection of entrance points which are provided by the *rnnode* function.

*Example 4.* An IS with a set of nodes  $\mathcal{V} = \{a, b, c, d, e, f, g\}$  is shown in Figure 3 (a). For the time being the reader can ignore the use of colors black and grey and consider the graph as a whole. There is a semantic index with two key concepts *a* and *c* pointing out to their respective nodes in the semantic network.

Similarly, the semantic network of Figure 2 has been converted to an IS by defining the index with four entries *P1* (page1.html), *P2* (page2.html), *vcard* and *vevent* and by removing the strong links. Thus, for instance, *vcard* entry points to the cycle of *vcard* nodes.

Given a graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  and two nodes  $v_1, v_n \in \mathcal{V}$ , if there is a sequence  $v_1, v_2, \dots, v_n$  of nodes in  $\mathcal{G}$  where  $(v_i, v_{i+1}) \in \mathcal{E}$  for  $1 \leq i \leq n-1$ , then we say that there is a *path* from  $v_1$  to  $v_n$  in  $\mathcal{G}$ . Given  $u, v \in \mathcal{V}$  we say that the node *v* is *reachable* from *u* if there is a path from *u* to *v*.

**Definition 4 (semantic sub-net).** Let  $IS = (\mathcal{V}, \mathcal{E}, \mathcal{I})$  be an indexed semantic network. Then, a semantic sub-net of *IS* with respect to concept *k* with  $(k, p) \in \mathcal{I}$  for some *p* is  $\mathcal{S}_k = (\mathcal{V}', \mathcal{E}')$  such that  $\mathcal{V}' = \{rnnode((\mathcal{V}, \mathcal{E}), k)\} \cup \{v | v \in \mathcal{V} \text{ and } v \text{ is reachable from } rnnode((\mathcal{V}, \mathcal{E}), k)\}$  and  $\mathcal{E}' = \{(u, v) | (u, v) \in \mathcal{E} \text{ and } u \in \mathcal{V}'\}$ .

*Example 5.* Figure 3 (a) shows in black color the semantic sub-net extracted from the whole IS with respect to concept *c*.

**Definition 5 (semantic relationship).** Given a semantic network  $\mathcal{S} = (\mathcal{V}, \mathcal{E})$  and a node  $v \in \mathcal{V}$ , the semantic relationships of *v* are the edges  $\{v \rightarrow v' \in \mathcal{E}\}$ . We say that a concept *v* is semantically related to a concept *u* if there exists a semantic relationship  $(u \rightarrow v)$ .

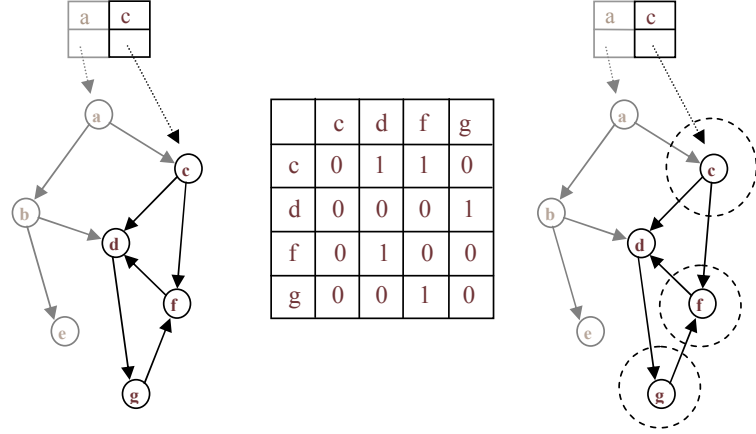
The semantic relations in our semantic networks are unidirectional. The semantics associated to the edges of a semantic network is not transitive because edges can have different meanings. Therefore, the semantic relation of Definition 5 is neither transitive.

Given a node *n* in a semantic network, we often use the term *semantically reachable* to denote the set of nodes which are reachable from *n* through semantic relationships. Clearly, semantic reachability is a transitive relation.

The following lemma ensures that an extracted sub-net does not change the semantics of its associated semantic network.

**Lemma 1.** Let *N* be the semantic sub-net extracted from the semantic indexed network  $IS = (\mathcal{V}, \mathcal{E}, \mathcal{I})$  with respect to concept *k*. Let  $n = rnnode((\mathcal{V}, \mathcal{E}), k)$ . Then *N* is formed by *n* and all and only the semantically reachable nodes from *n*, and all and only the semantic relationships of its nodes.

*Proof.* The claim trivially holds from the fact that *N* is a subset of *IS*, i.e., *N* does not add new nodes nor edges to the semantic network; and also, the nodes and edges of *N* are all those nodes and edges in all the paths starting at the node  $n = rnnode((\mathcal{V}, \mathcal{E}), k)$ . Therefore, *n* and all the semantically reachable nodes from *n* belong to *N*; and all the semantic relationships of *n* and the nodes in the paths are preserved because the paths are only traversed forwards.



**Fig. 3.** a) A semantic sub-net. b) The sub-net's adjacency matrix. c) A backward slice.

## 4.2 Semantic sub-net slicing

In this section we present a procedure that allows us to extract a portion of a semantic sub-net according to some criterion. The procedure uses an *adjacency matrix* to represent the semantic sub-net.

The adjacency matrix  $m$  of a directed graph  $\mathcal{G}$  with  $n$  nodes is the  $n \times n$  matrix where the non-diagonal entry  $m_{ij}$  contains 1 if there is an edge such that  $m_i \rightarrow m_j$ .<sup>3</sup>

*Example 6.* Consider the semantic sub-net in Figure 3 (a). Node c has two directed edges, one to node d and other to node f. Thus, in the entry  $m_{cd}$  and  $m_{cf}$  we write 1, and 0 in the other cells.

Now, we are in a position to introduce our slicing based method for information recovering from semantic sub-nets. In traditional program slicing, the user selects a variable in a sentence of a program, and the slicer extracts the part of the program that has an influence over this variable. This can be done thanks to the use of a *Program Dependence Graph* [12] that stores the control and data dependences in a program. In our context, the slicing criterion is different to the standard program slicing technique which consists in a single point. Thanks to the introduction of indexes we can enrich our notion of slicing criterion by adding an extra level of information which allows us to perform slicing at two different levels. Firstly, we can select a concept in the index. From this concept we can extract a semantic sub-net as described before. Next, in the resultant semantic subnet we can select the node of interest. Hence, our slicing criterion consists of a pair formed by a key concept and a node. Formally:

<sup>3</sup> Note that we could write a label associated to the edge in the matrix instead of 1 in order to also consider other relationships between nodes.

**Definition 6 (slicing criterion).** Let  $IS = (\mathcal{V}, \mathcal{E}, \mathcal{I})$  be an indexed semantic network. Then a slicing criterion  $\mathcal{C}$  for  $IS$  is a pair of elements  $\langle k, v \rangle$  such that  $(k, p) \in \mathcal{I}$  for some  $p$ ,  $v \in \mathcal{V}'$  and  $S_k = (\mathcal{V}', \mathcal{E}')$  is the semantic sub-net of  $IS$  with respect to concept  $k$ .

Intuitively, the slicing criterion contains the concept of interest, from which we can extract a relevant sub-net, and a single node of the computed sub-net. This node is a particular microformatting class with the semantic information of interest reachable through semantic relations. Given a semantic sub-net, we can produce two different slices by traversing the sub-net either forwards or backwards from the node pointed out by the slicing criterion. Each slice gives rise to different semantic information.

*Example 7.* Consider the slicing criterion  $\langle c, d \rangle$  for the IS in Figure 3 c). The first level of slicing uses  $c$  to extract the semantic subnet highlighted with black color. Then, the second level of slicing performs a traversal of the semantic sub-net either forwards or backwards from  $d$ . In Figure 3 c) the backward slice contains all nodes whereas the forward slice would only contain  $\{d, f, g\}$ .

Both backward slicing [8] and forward slicing [18] are well-known and widely used techniques in the literature (see, e.g., [10, 19, 20]). In our context, they can be used to distinguish between two different semantic relations: While backward slicing produces more general information (i.e., the classes to which the slicing criterion belongs), forward slicing produces specialized semantic relations (i.e., the classes which belong to the slicing criterion).

*Example 8.* Consider the semantic network in Figure 2 together with the slicing criterion  $\langle P1, adr:P1 \rangle$ . With  $P1$  we can perform the first level of slicing to recover a semantic sub-net which is composed by the nodes  $\{P1, vcard:P1, vcard:P2\}$  and all of their descendant (semantically reachable) nodes. Then, from node  $adr:P1$  we can go forwards and collect the information related to the address or backwards and collect nodes  $vcard:P1$ ,  $P1$  and  $vcard:P2$ . The backward slicing illustrates that the node  $adr:P1$  is semantically reachable from  $P1$ ,  $vcard:P1$ , and  $vcard:P2$ , and thus, there are semantic relationships between them. Hence, we extract a slice from the semantic network and, as a consequence, from the semantic web.

We can now formalize the notion of forward/backward slice for semantic sub-nets. In the definition we use  $\rightarrow^*$  to denote the reflexive transitive closure of  $\rightarrow$ .

**Definition 7 (forward/backward slice).** Let  $IS = (\mathcal{V}, \mathcal{E}, \mathcal{I})$  be an indexed semantic network with  $(k, p) \in \mathcal{I}$  for some  $p$ . Let  $S_k = (\mathcal{V}', \mathcal{E}')$  be the semantic sub-net of  $IS$  with respect to  $k$  and  $\mathcal{C} = \langle k, node \rangle$  a slicing criterion for  $IS$ . Then a slice of  $IS$  is  $\mathcal{S}' = (\mathcal{V}_1, \mathcal{E}_1)$  such that

**forward**  $\mathcal{V}_1 = \{node\} \cup \{v | v \in \mathcal{V}' \text{ and } (node \rightarrow^* v) \in \mathcal{E}'\}$   
**backward**  $\mathcal{V}_1 = \{node\} \cup \{v | v \in \mathcal{V}' \text{ and } (v \rightarrow^* node) \in \mathcal{E}'\}$

---

**Input:** An indexed semantic network  $IS = (\mathcal{V}, \mathcal{E}, \mathcal{I})$   
 and a slicing criterion  $\mathcal{C} = \langle k, node \rangle$  where  $(k, p) \in \mathcal{I}$  for some  $p$   
**Output:** A slice  $\mathcal{S}' = (\mathcal{V}', \mathcal{E}')$   
**Initialization:**  $\mathcal{V}' := \{node\}, \mathcal{E}' := \{\}, Visited := \{\}$   
**Begin**  
 Compute  $S_k = (\mathcal{V}_k, \mathcal{E}_k)$  a semantic sub-net of  $IS$   
 whose adjacency matrix is  $\mathcal{M}$   
**Repeat**  
   **let**  $s \in (\mathcal{V}' \setminus Visited)$   
   **let**  $c := column(s, \mathcal{M})$   
   **For each**  $s' \in \mathcal{V}_k$  with  $r = row(s', \mathcal{M})$  and  $\mathcal{M}_{r,c} = 1$   
      $\mathcal{V}' := \mathcal{V}' \cup \{s'\}$   
      $\mathcal{E}' := \mathcal{E}' \cup \{(s' \rightarrow s)\}$   
      $Visited := Visited \cup \{s\}$   
**Until**  $\mathcal{V}' = Visited$   
**End**  
**Return:**  $(\mathcal{V}', \mathcal{E}')$

---

**Fig. 4.** An algorithm for semantic network backward slicing.

and  $\mathcal{E}_1 = \{(u \rightarrow v) \mid (u \rightarrow v) \in \mathcal{E}' \text{ with } u, v \in \mathcal{V}_1\}$

The algorithm of Figure 4 shows the complete slicing based method for information extraction from semantic networks. Roughly speaking, given an IS and a slicing criterion, (i) it extracts the associated semantic sub-net, (ii) it computes the sub-net's adjacency matrix, and (iii) it extracts (guided by the adjacency matrix) the nodes and edges that form the final slice.

The algorithm uses two functions  $row(s, \mathcal{M})$  and  $column(s, \mathcal{M})$  which respectively return the number of row and column of concept  $s$  in matrix  $\mathcal{M}$ . It proceeds as follows: Firstly, the semantic sub-net associated to  $IS$  and the adjacency matrix of the sub-net are computed. Then, the matrix is traversed to compute the slice by exploiting the fact that a cell  $\mathcal{M}_{i,j}$  with value 1 in the matrix means that the concept in column  $j$  is semantically related to the concept in row  $i$ . Therefore, edges are traversed backwards by taking a concept in a column and collecting all concepts of the rows that have a 1 in that column.

Now, we present the main result of the paper which states that the slicing method is correct with respect to semantic relationships.

**Theorem 1.** *Let  $IS = (\mathcal{V}, \mathcal{E}, \mathcal{I})$  be an indexed semantic network,  $\langle k, node \rangle$  a slicing criterion such that  $n = rnode((\mathcal{V}, \mathcal{E}), k)$ , and  $\mathcal{S}'$  the backward slice returned by the semantic network backward slicing algorithm. Then,  $\mathcal{S}'$  is formed by  $node$  and all the nodes from which  $node$  is semantically reachable in the semantic sub-net induced by  $k$ . Moreover, all and only the semantic relationships of the nodes in  $\mathcal{S}'$  that appear in  $IS$  also appear in  $\mathcal{S}'$ .*

*Proof.* Firstly,  $\mathcal{S}'$  is extracted from the semantic sub-net  $S_k = (\mathcal{V}', \mathcal{E}')$  computed with respect to  $k$  with  $rnode((\mathcal{V}, \mathcal{E}), k)$ . Moreover, by Lemma 1 we know that all

the nodes in  $S_k$  are nodes of  $IS$  and they all keep their semantic relationships. In addition, we know by Definition 6 that  $n$  belongs to  $\mathcal{V}'$ . Then, since the algorithm only collects nodes which are transitively connected to  $n$ , we can ensure that  $node$  and all the nodes from which it is semantically reachable are in  $S'$ . Moreover, all edges in the paths also belong to the slice, and hence, all the semantic relationships of the nodes in  $S'$  that appear in  $IS$  also appear in  $S'$ . Furthermore,  $S'$  only collects the relations participating in the paths from  $node$  and thus only the semantic relationships of the nodes in  $S'$  that appear in  $IS$  also appear in  $S'$ .

*Ongoing practical approach:* In order to demonstrate the usefulness of our approach we have implemented some tools for discovering and extracting the semantic relationships among web pages. The current prototype is able to analyse a complete web site by traversing its hyperlinks and identifying semantic relations between web pages. As an example, Table 1 shows the collection of microformats found in a set of web pages. Concretely, we launched several queries to the Google web search engine and took the first eight links for each query. The tool automatically analysed Google's results, and it found out their microformats and the semantic relations between the web pages. Certainly, there are notable efforts to extract microformats from web pages [21], and to filter HTML documents [22]; however current approaches only focus on single web pages, and thus, they ignore the relations between data which is located in different web pages.

**Table 1.** Searching for Microformats

Google query	web pages	vevent	vcard	geo	hresume	hreview
event sport upcoming "New York"	8	17	2	18	0	0
restaurant Paris food	8	0	21	6	0	0
"medical services" madrid hospital	8	0	20	0	0	0
hotel London quality room downtown	8	0	0	17	0	0
personal service "Los Angeles" street	8	0	15	2	0	0
song author	8	0	13	0	0	0
<b>Totals</b>	<b>48</b>	<b>17</b>	<b>71</b>	<b>37</b>	<b>0</b>	<b>0</b>

## 5 Related work and conclusions

In [23], three prototype hypertext systems were designed and implemented. In the first prototype, an unstructured semantic net is exploited and an authoring tool is provided. The prototype uses a knowledge-based traversal algorithm to facilitate document reorganization. This kind of traversing algorithms is based on typical solutions like depth-first search and breadth-first search. In contrast, our IS allow us to optimize the task of information retrieval.

[24] designed a particular form of a graph to represent questions and answers. These graphs are built according to the question and answer requirements. This is in some way related to our work if we assume that our questions are the slicing criteria and our answers are the computed slices. In our approach, we conserve

a general form of semantic network, which is enriched by the index, so, it still permits to represent sub-graphs of knowledge.

To the best of our knowledge this is the first program slicing based approach to extract information from the semantic web. The obtained answers are semantically correct, since, the information extraction method follows the paths of the source semantic tree, i.e., the original semantic relationships are preserved. Furthermore, semantic relationships contained in sets of microformatted web pages can also be discovered and extracted.

Program slicing has been previously applied to data structures. For instance, Silva [10] used program slicing for information extraction from individual XML documents. He also used a graph-like data structure to represent the documents. However semantic networks are a much more general structure, that could contain many subgraphs, while XML documents are always a tree-like structure. In contrast to this method, our approach can process groups of web pages.

This method could be exploited by tools that feed microformats. Frequently, these tools take all the microformats in the semantic web and store them in their databases in order to perform queries. Our representation improves this behavior by allowing the system to determine what microformats are relevant and what microformats can be discarded. Another potential use is related to automatic information retrieval from websites by summarizing semantic content related to a slicing criterion. Similarly, web search engines could use this method to be able to establish semantic relations between unrelated links.

To summarize, we have introduced an approach for information extraction from the semantic web. This approach is based on program slicing, and has many potential applications in the design of modern tools for information extraction.

## References

1. J. Hendler T. Berners-Lee and O. Lassila. The Semantic Web. *Scientific American Magazine*, May 2001.
2. T. Çelik. What's the Next Big Thing on the Web? It May Be a Small, Simple Thing - Microformats. *Knowledge@Wharton*, 2005.
3. Microformats.org. The Official Microformats Site. <http://microformats.org/>, 2009.
4. R. Khare and T. Çelik. Microformats: a Pragmatic Path to the Semantic Web. In *WWW '06: Proceedings of the 15th International Conference on World Wide Web*, pages 865–866. ACM, 2006.
5. hCard. Simple, Open, Distributed Format for Representing People, Companies, Organizations, and Places. <http://microformats.org/wiki/hcard>, 2009.
6. R. Khare. Microformats: The Next (Small) Thing on the Semantic Web? *IEEE Internet Computing*, 10(1):68–75, 2006.
7. J. F. Sowa. Semantic Networks. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*. John Wiley & Sons, 1992.
8. M. Weiser. Program Slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, 1984.
9. C. Ochoa, J. Silva, and G. Vidal. Lightweight Program Specialization via Dynamic Slicing. In *Proc. of the Workshop on Curry and Functional Logic Programming (WCFLP 2005)*, pages 1–7. ACM Press, 2005.

10. J. Silva. A Program Slicing Based Method to Filter XML/DTD Documents. In *SOFSEM (1)*, pages 771–782, 2007.
11. F. Tip. A Survey of Program Slicing Techniques. *Journal of Programming Languages*, 3:121–189, 1995.
12. J. Ferrante, K. J. Ottenstein, and J.D. Warren. The Program Dependence Graph and Its Use in Optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 9(3):319–349, 1987.
13. R. Quillian. Semantic Memory. In Marvin Minsky, editor, *Semantic Information Processing*. MIT Press, 1969.
14. Gerd Stumme, Bettina Hoser, Christoph Schmitz, and Harith Alani, editors. *ISWC 2005 Workshop on Semantic Network Analysis*, volume 171 of *CEUR Workshop Proceedings*, Galway, Ireland, 2005.
15. Harith Alani, Bettina Hoser, Christoph Schmitz, and Gerd Stumme, editors. *Proceedings of the 2nd Workshop on Semantic Network Analysis*, 2006.
16. J. F. Sowa, editor. *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, 1991.
17. hCalendar. Simple, Open, Distributed Calendaring and Events Format. <http://microformats.org/wiki/hcalendar>, February 2009.
18. J.F. Bergeretti and B. Carré. Information-Flow and Data-Flow Analysis of while-Programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1):37–61, 1985.
19. J. Silva and G. Vidal. Forward Slicing of Functional Logic Programs by Partial Evaluation. *Theory and Practice of Logic Programming*, 7:215–247, 2007.
20. C. Ochoa, J. Silva, and G. Vidal. Dynamic Slicing Based on Redex Trails. In *Proc. of the ACM SIGPLAN 2004 Symposium on Partial Evaluation and Program Manipulation (PEPM'04)*, pages 123–134. ACM Press, 2004.
21. C. Yu. Tails add-on. Available at: <http://blog.codeeg.com/tails-firefox-extension-03/>, 2007.
22. J. Silva. Web filtering toolbar 1.3. Available at: <https://addons.mozilla.org/es-ES/firefox/addon/5823>, 2008.
23. W. Wang and R. Rada. Structured Hypertext with Domain Semantics. *ACM Transactions on Information Systems (TOIS)*, 16(4):372–412, 1998.
24. D. Mollá. Learning of Graph-based Question Answering Rules. In *Proc. HLT/NAACL 2006 Workshop on Graph Algorithms for Natural Language Processing*, pages 37–44, 2006.





## Author Index

Almendros-Jimenez, Jesus M. ....	69
Arroyo, Gustavo ....	145
Banti, Federico ....	21
Berczes, Tamas ....	37
Bry, François ....	3
Dueñas, Juan C. ....	99
Fardal, Frank ....	115
Freitag, Burkhard ....	53
García, Boni ....	99
Guta, Gabor ....	37
Jaksic, Mirjana ....	53
Kusper, Gabor ....	37
Linaje, Marino ....	85
Lozano-Tello, Adolfo ....	85
Lucas, Francisco J. ....	129
Malecha, Gregory ....	5
Molina, Fernando ....	129
Morrisett, Greg ....	5
Nietzio, Annika ....	115
Olsen, Morten Goodwin ....	115
Parada G., Hugo A. ....	99
Polleres, Axel ....	1
Preciado, Juan Carlos ....	85
Pugliese, Rosario ....	21
Ramos, J. Guadalupe ....	145
Rodríguez, Roberto ....	85
Sanchez-Figueroa, Fernando ....	85
Schönberg, Christian ....	53
Schreiner, Wolfgang ....	37
Silva, Josep ....	145
Snaprud, Mikael ....	115
Solorio, Juan Carlos ....	145
Sztrik, Janos ....	37
Tiezzi, Francesco ....	21
Toval, Ambrosio ....	129
Weitl, Franz ....	53
Wisnesky, Ryan ....	5