Qualifying Semantic graphs using Model Checking

Mahdi Gueffaz, Sylvain Rampacek, Christophe Nicolle

Qualifying Semantic graphs using Model Checking

Mahdi Gueffaz, Sylvain Rampacek, Christophe Nicolle
LE2I, UMR CNRS 5158
University of Bourgogne
BP 47870, 21078 Dijon Cedex, France
{Mahdi.Gueffaz, Sylvain.Rampacek, Chistophe.Nicolle}@u-bourgogne.fr

***Abstract*— Semantic interoperability problems have found their solutions using languages and techniques from the Semantic Web. The proliferation of ontologies and meta-information has improved the understanding of information and the relevance of search engine responses. However, the construction of semantic graphs is a source of numerous errors of interpretation or modeling and scalability remains a major problem. The processing of large semantic graphs is a limit to the use of semantics in current information systems. The work presented in this paper is part of a new research at the border of two areas: the semantic web and the model checking. This line of research concerns the adaptation of model checking techniques to semantic graphs. In this paper, we present a first method of converting RDF graphs into NµSMV and PROMELA languages.**

Keywords: Semantic graph; Model-checking; temporal logic.

I. INTRODUCTION

W3C [1] aims to standardize the representation and the exchange of information on the WEB. This objective should help make the information understandable for both automated processes and users. The homogenization of computer exchanges took place due to the introduction of the XML [1] standard. This standard has enabled the program to manipulate information through languages with hierarchical structure mark-up defined by grammars derived from the XML standard. However, this effort has not helped improve the user's understanding of information. Thus, new standards have been developed to enable the semantic representation of information in the form of XML-derived languages. This base is called Semantic Web standards and it is usually represented as a stack of languages ranging from automatic processes oriented languages to languages representing more abstract concepts of formal semantics [2]. These languages are used to represent the semantics associated with information, whatever its form and structure. To allow the construction of semantic graph, many tools have been developed like Annotea [3] which is a project of the W3C that specifies the infrastructure for the annotation of Web documents. The main format used in the annotation is RDF and the types of documents can be annotated are HTML documents or XML based. However, none provides the functionality to verify the consistency of semantics, and reduce errors annotations.

This paper proposes a new way to check these semantic graphs by model-checking in order to reduce errors in annotation and make the data more relevant. Model checking is an automatic verification technique, it has been applied to many cases in industry, for example [4], in the Netherlands, model-checking has revealed several serious flaws in the design of control system of a barrier protection against flooding which protects the main port of Rotterdam against floods. The large manufacturing company processor "Intel" has used the model-checking to detect the bug in its Pentium II processor that caused a loss of 475 million dollars damage to the reputation of INTEL. Finally, the model-checking was allowed to find an error in the system of handling baggage at the Denver airport (USA), delayed opening its doors for 9 months and a loss of 1.1 million dollars per day.

Model checking is a powerful tool for system verification because it can reveal errors that were not discovered by other formal methods such as testing or simulation. Model checking uses temporal logic to describe the properties checking the system model. As we have seen in the examples above, the model-checking can handle complex problems with large amounts of information, stored as a graph, in order to verify critical systems. In comparison, in the semantic web, the use of graphs is pervasive and serious problems of scalability arise [5]. Thus, it is appropriate to use the algorithms developed for model-checking to the field of Semantic Web.

Related Work

In this section, we briefly discuss some of the researches related to the verification of the Semantic graphs using the model checking. There are very few researches about the use of the model checking method to qualify a Semantic graph. On the contrary, there are several researches that verify the Web application. The work in [6] proposes a new way of converting an RDF graph to the BCG[2] format that was used in the CADP[3] toolbox.

The CADP toolbox is a verification toolbox for asynchronous concurrent systems. The toolbox accepts as input several languages and all of these languages are compiled into LTS [4], which are state/transition graphs representing the behavior of concurrent systems. CADP provides several representations for LTS; one of these representations is the BCG format.

There are several researches in which the Web application is modeled as a directed graph. In [7], [8], the components of a window (e.g., a page, a frame and a link) are modeled as

[1] World Wide Web Consortium

[2] Binary Coded Graph
[3] Construction and Analysis of Distributed Processes
[4] Labeled Transition System

states. The reachability between states is defined as requirements of the Web application and they are verified using the model checking. In [9], [10], [11], they took an approach to model the Web applications using parallel composition of the UML diagrams. The work in [12] proposes the way to discriminate states of interest by introducing a specialized operator for LTL. They use it to verify the Web applications.

In [13], the authors propose a behavioral model of the Web application, called 'Web Automata' based on the MVC[5] model architecture. They model the behavior of a Web application with dynamic contents as an extension of links-automata with the constraint-logic feature of the Extended Finite Automata (EFA). The testing framework of Web applications based on the behavioral model is also presented in their research. In [14], the authors present a formal approach for modeling Web applications using communicating automata. They observe the external behavior of an explored part of a Web application using a monitoring tool. The observed behavior is then converted into communicating automata representing all windows, frames and framesets of the application under test by intercepting HTTP[6] requests and responses using a proxy server.

II. MODEL CHECKING AND TEMPORAL LOGIC OVERVIEW

Formal methods [15] offer great potential for an early inclusion of verification in the design process, providing technical audit more efficiently and reduce the verification time. Formal methods are highly recommended techniques for the development of software. They have led to the development of some very promising verification techniques that facilitate early detection of defects. Two types of formal verification methods can be distinguished: methods based on the proof of the theorem and the methods based on models.

Methods based on the proof of the theorem verify the correctness of systems by properties in a mathematical theory. These properties are proven with the highest possible precision using tools such as theorem provers and proof checkers. Theorems proof are also called proof assistants.

Methods based on models describe the possible system behavior in a mathematical precise and unambiguous manner. The system models are accompanied by algorithms that systematically explore all states of the system model. This provides the basis for a whole range of verification techniques ranging from an exhaustive exploration "Model checking" to experiments with a restrictive set of scenario in the model "Simulation". Simulation allows the user to study the system behavior. It is less suited to detect errors because it is difficult to generate all possible scenarios of the system and to simulate them all. Model checker is a verification technique that explores all possible system state. In this way, it can be shown that a given system model truly satisfies a certain property.

The model checker examines all relevant system states in order to check whether they satisfy the desired property. The model checker gives a counter example that indicates how the model can violate the property. With a help of a simulator, the user can locate the error and adapt the model or the property to prevent the violation of property (Fig. 1).

[5] Model, View, and Control
[6] Hypertext Transfer Protocol

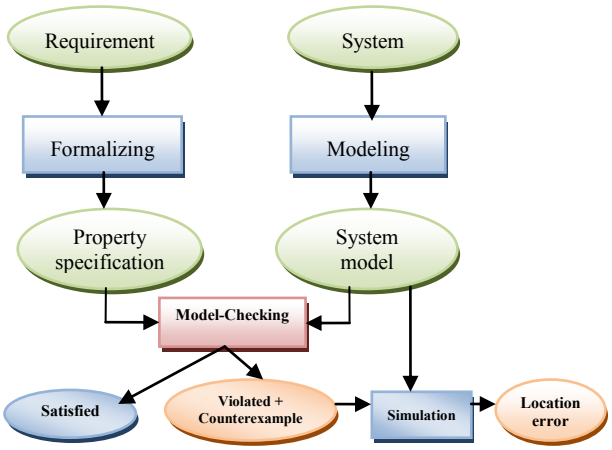

Figure 1. The model checking approach.

For our approach we will use model checking to analyze semantic networks. We use both linear time logic "LTL" and computation tree logic "CTL" for describing the specifications of the properties to be verified with model checking.

Algorithm: Model-checking
Begin
While stack ≠ nil ***do***
P := top (stack);
while ¬ satisfied (p) ***then***
Refine the model, or property;

Else if satisfied (p) ***then***
P := top (stack);

Else *// out of memory*
Try to reduce the model;
End
End

The concepts of temporal logic used for the first time by Pnueli [16] in the specification of formal properties are fairly easy to use. The operators are very close in terms of natural language. The formalization in temporal logic is simple enough although this apparent simplicity therefore requires significant expertise. Temporal logic allows representing and reasoning about certain properties of the system, so it is well-suited for the systems verification. There are two main temporal logics, that is linear time and branching time. In linear time temporal logic, each execution of the system is independently analyzed. In this case, a system satisfies a formula f, if f holds along every execution. The branching time combines all possible executions of the system into a single tree. Each path in the tree is a possible representation of the system execution.

III. THE SCALESEM APPROACH

This section details our approach which consists in transforming semantic graphs into models in order to be verified by the model-checker. For this, we have developed two tools called "RDF2SPIN" and "RDF2NμSMV", that

transform semantic graphs into PROMELA [7] and respectively into NμSMV [17] language.

We use SPIN [18] and NμSMV as model checkers to check the model of semantic graphs. We want to compare them in term of capabilities. SPIN is a software tool for verifying system models. The system is described in a language model called PROMELA. NμSMV is the amelioration of SMV model checker, working on the same simple principles as SMV. SPIN verifies the correctness of properties expressed in linear time logic; on the other side NμSMV verifies the properties in both linear time logic and computation tree logic.

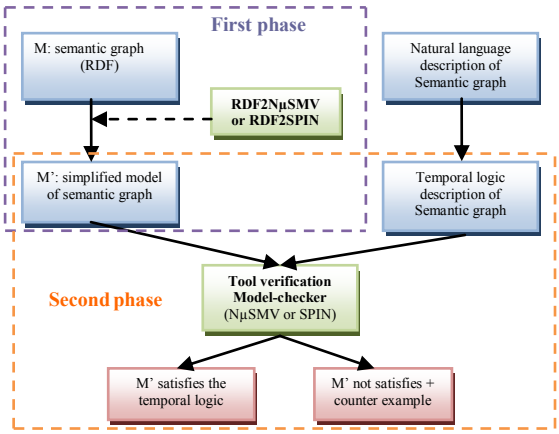

Figure 2. The ScaleSem architecture.

In Fig. 2, we present the architecture of our approach. In this architecture, from a natural language description, we can get the semantic graph (RDF[8]) and its description in temporal logic, as shown in the example found in the section VII. We divide this architecture in two phases. The first phase concerns the transformation of the semantic graph into a model using our tools RDF2SPIN and RDF2NμSMV. There are three steps in this transformation. The first step is to explore the entire RDF graph to obtain the triplets table. The second step is to determine a root for the graph, and the last step is to write the model that represents the semantic graph in the PROMELA or NμSMV languages. The second phase concerns the verification of properties expressed in temporal logic on the model using the model-checker SPIN or NμSMV. The choice of model checker depends on the tool that you use to convert the semantic graphs. For example, when using RDF2SPIN, you must use the model-checker SPIN to check your model.

A. *Introducing RDF*

RDF is a language developed by the W3C to bring a semantic layer to the Web [19]. It allows the connection of Web resources using directed labeled edges. The structure of RDF documents is a complex labeled directed graph. An RDF document is a set of triples <subject, predicate, object>. In addition, the predicate (also called property) connects the subject (resource) to the object (value). Thus, the subject and the object are nodes of the graph connected by an edge directed from the subject towards the object. The nodes and the edges belong to "resource" types. A resource is identified by a Uniform Resource Identifier [20].

The declarations can also be represented as a graph, the nodes as resources and values, and arcs as properties. The resources are represented in the graph by circles; the properties are represented by directed arcs and values by a box (a rectangle). Values can be resources if they are described by additional properties. For example, when a value is a resource in another triplet, the value is represented by a circle [21].

The RDF graphs considered here are represented as XML verbose files, in which the information is not stored hierarchically (so-called graph point of view). These RDF graphs, that is, in fact, the semantic model of an RDF file, are not necessarily connected, meaning they may have no root vertex from which all the other vertices are reachable. The RDF graph transformation into a model is articulated in three steps: exploring the RDF graph, holding election of the root vertex, generating the model of the semantic graph.

B. *Exploring RDF graph*

In order to exploit the RDF graphs by using SPIN or NμSMV, we therefore have to determine whether they have a root vertex, by analyzing RDF triples, and if this is not the case, we must create a new root vertex by taking care to keep the size of the resulting graph as small as possible.

We achieve this by appropriate explorations of the RDF graphs, as explained below. Let us consider that an RDF graph is represented as a couple (V, E), where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. For a vertex x, we note $E(x) = \{ y \in V \mid (x, y) \in E \}$ the set of its successor vertices. This corresponds to the classical data structure for representing graphs in memory, consisting of an array indexed by the vertices and containing in each entry the list of successor vertices of the corresponding vertex. There are several algorithms to traverse a large graph, of these basic algorithms include the best known, depth-first search (DFS) and breadth-first search (BFS). We use depth-first search algorithm, illustrated below to explore graph, knowing that the breadth-first algorithm also works in this context. We considered here an iterative variant of DFS which makes use of an explicit stack, rather than the recursive variant given in [22]; this is required in practice to avoid overflows of the system call stack when the algorithm is invoked for exploring large graphs.

```
Algorithm: procedure Dfs (x):
begin
  visited(x) :=  true;
   // vertex x becomes visited
  p(x) := 0; // start exploring its successors
  stack := push(x, nil);
 while stack ≠ nil do
y := top(stack);
if p(y) < |E (y)| then
 // y has some unexplored successors
 z := E (y) p(y);
    p(y) := p(y)+1;
    // take the next successor of y
    if ¬visited (z) then
    visited(z) := true;  // visit it
    p(z) := 0;//start exploring its successors
```

[7] Process Meta Language
[8] Resource Description Framework

```
    stack := push(z, stack)
    endif
   else //all successors of y were explored
   stack := pop(stack)
  endif
 end
end
```

C. Determining a Root Vertex

If the RDF graph has no vertex root, we must create a root as to be the successors of all vertices of the graph but it will increase the number of edges. We look forward to doing this by adding a few edges as possible. A vertex x of a directed graph is a partial root if it cannot be reached from any other vertex of the graph. If the graph contains only one partial root, all other vertices of the graph can be reached from the root, otherwise there would be other roots in the partial graph. If the graph has multiple partial roots, the most economical way to provide a root is to create a new record with all the roots as a partial successor: this will add to the graph a minimum number of edges. We compute the set of partial roots in two phases, each one consisting in successive explorations of the graph. The first phase identifies a set of candidate partial roots, and the second one refines this set in order to determine the partial roots of the graph.

Remark: a property must always have a resource and a value; the resource should never be a value with the same predicate, i.e. a loop in the graph.

```
Algorithm: procedure RootElection():  //
precondition: ∀ x ∈ V.visited(x) = false
Begin // first phase
 root_list := nil;
 forall  x ∈ V do
  if ¬visited(x) then
   Dfs(x);
   root_list := cons(x, root_list)
  endif
 endfor;
//second phase
 if |root_list|= 1 then
  root := head(root_list)
  // the single partial root is the global root
  else
  forall x ∈ V do  visited(x):= false;
  endfor;
  forall x ∈ root_list do
   // reexplore partial roots in reverse order
   if ¬visited(x) then Dfs(x)
   else
    root_list := root_list \ {x}
     // partial root is not a real one
   endif
  endfor;
  if |root_list| = 1 then
   root := head(root_list)
   // a single partial root is the global root
   else
   root := new_node();
   // new root predecessor of the partial roots
   E(root) := root_list
  endif
 endif
```

The first phase explores the graph until it is fully explored, and inserts in root_list all vertices that have no predecessor. If root_list contains a single vertex, so overall it is the global root of the graph since all the other vertex are accessible from it and it is useless to the second phase has passed. Otherwise, any vertex contained in root_list could also be a root of the graph: the role of the second phase is to determine which of the partial root the root of the global graph is.

The second phase performs a new wave of exploration of the roots contained in partial root_list in reverse order in which they were inserted in the list. If a root in root_list is to be visited by a partial root, it is removed from the list because it is not a partial root. At the end of this phase, all partial roots of the graph are present in root_list. Indeed, each vertex is unreachable from the partial roots which were explored during the second phase. A new root is created (see Fig. 3), having as successor all the partial roots of root_list, which ensures that all vertices of the graph are accessible from the new root. Therefore, such a summit is inaccessible from other nodes of the graph.

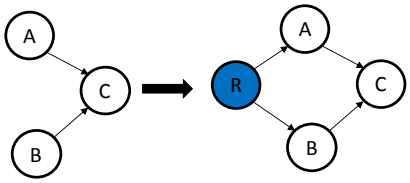

Figure 3. A root is a single node that has no predecessor. In this graph, we have node A and node B, two roots, and then we will create a new virtual root (blue circle "R") that points to the two roots.

The algorithm for determining a root has a complexity O(|V|+|E|), linear in the size of the graph (number of vertices and edges), since each phase visits every state and traverses every edge of the graph only once. Given that the graph must be traversed entirely in order to determine whether it has a root or not, this complexity is optimal.

D. Generating the model

The third step is divided into three sub-steps. The first one consists in creating the table of all triples by exploring the entire graph; the second one consists in generating the table of resources and values for RDF2SPIN but for RDF2NμSMV, it generates the table of association. The last one consists in producing the model representing the semantic graph written in PROMELA or in NμSMV language.

Table of triples - We will create a table consisting in resources, properties and values, by exploring the RDF graph. In our RDF graph, the resource and the value are represented by nodes and the property is an edge directed from the resource towards the value. The table of triples of the RDF graph is useful for the next sub-step.

In this second sub-step, RDF2SPIN generates a table of resources and values, while RDF2NμSMV generates a table of association.

- *Table of resources and values* - Browsing the table triples seen in the previous step, we attribute a unique function for each resource and for each value. These functions are of proctype type. We combine all these functions in a table called table of resources and values.
- *Table of association* – This table contains an identifier for each resource, property and value.

The model - In this last sub-step, we will write the model in PROMELA language for RDF2SPIN tool or in NμSMV language for RDF2NμSMV tool, corresponding to the RDF graph that we want to check.

IV. EXAMPLE AND BENCHMARK

To illustrate our approach, we take the natural language description in [23] as follows:

Ninety-three is a novel by Victor Hugo published in 1874, whose theme is the French Revolution. Victor Hugo was born in February 26, 1802 in Besançon.

From the description above, we can easily extract simple propositions, see Table 1 described below, and also the proposal described in temporal logic a little lower. Table 2 presents the RDF triples derived from the Table 1.

TABLE 1. SHORT LIST OF SIMPLE PROPOSITION

1	"Ninety-three is a novel"
2	"Ninety-three its author is Victor Hugo"
3	"Ninety-three has been published in 1874"
4	"Ninety-three's theme is the French revolution"
5	"Victor Hugo was born in February 26, 1802"
6	"Victor Hugo was born in Besançon"

TABLE 2. CORRESPONDING RDF TRIPLES

	Subject	Predicate	Object
1	Ninety-three	is	Novel
2	Ninety-three	author	Victor Hugo
3	Ninety-three	Published	1874
4	Ninety-three	theme	French revolution
5	Victor Hugo	Date_born	February 26, 1802
6	Victor Hugo	Place_born	Besançon

From the previous description in natural language, we can express it in temporal logic as shown in Table 3. These temporal logics are expressed in a general way, but they can be expressed as well in both linear time logic and computation tree logic.

TABLE 3. EXAMPLE OF TEMPORAL LOGIC REPRESENTATION

	Temporal logic	Explanation
1	Always (Ninety-three → next novel)	We check that ninety three is a novel
2	Always (Ninety-three → next Victor Hugo)	We check that ninety three is written by Victor Hugo
3	Always (Ninety-three → next 1874)	We check that ninety three is published in 1874
4	Always (Ninety-three → next French revolution)	We check that ninety three' them is the French revolution
5	Always (Victor Hugo → next 26 February 1802)	We check that Victor Hugo was born in 26 February 1802
6	Always (Victor Hugo → next Besançon)	We check that Victor Hugo was born in Besançon

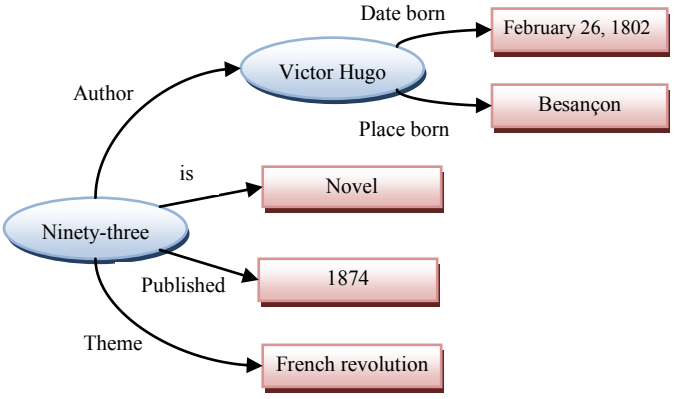

Figure 4. RDF graph.

Now, we will be able to transform the RDF graph in Fig. 4 with our tools "RDF2SPIN" and RDF2NμSMV" into a model in order to check each formula of temporal logic described in Table 3 and see if each formula is verified or not in the model with the SPIN and NμSMV model-checkers. In this way we can verify the semantic graphs.

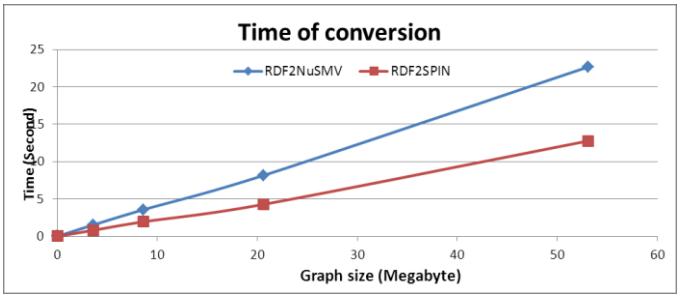

Figure 5. Time of conversion of Semantic graphs.

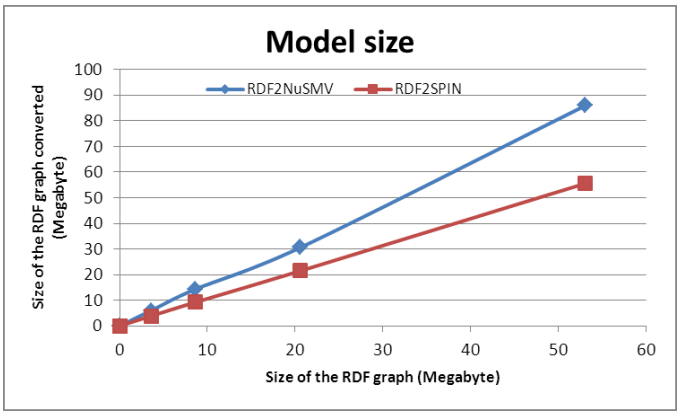

Figure 6. Size of the models.

We tested our tools on several RDF graphs, and we calculated the time of conversion as shown in Fig 5. Note that RDF2SPIN tool is faster in converting semantic graphs than the RDF2NμSMV tool. Both tools are quick in converting semantic graphs; we have less than 15 seconds for a graph of 53 MB size with RDF2SPIN tool and almost 21 seconds with RDF2NμSMV tool. Both transformation tools follow a polynomial curve. In Fig. 6, we see the size of converted semantic graphs from RDF to PROMELA language with RDF2SPIN and NμSMV language with RDF2NμSMV. We remark that the sizes of PROMELA model are smaller than the NμSMV model.

V. CONCLUSION

This paper presents a new technique for the semantic graphs verification by using a model-checker. Knowing that the model-checker does not understand the semantic graphs, we developed two tools RDF2SPIN and RDF2NμSMV to convert them into PROMELA and NμSMV languages in order to be verified with the temporal logics. There are formulas that can be presented in LTL and not in CTL and vice versa. The advantage of NμSMV is that the verification can be made with both linear time logic and computation tree logic formulas.

In future work, we would like to convert the SPARQL query language for RDF graphs into queries using the operator of the temporal logic, to have a better verification of RDF graphs representing the building industry.

REFERENCE

[1] T. Bray, J. Paoli, C. Sperberg-McQueen. M., Maler, E., Yergeau, F., Cowan, J.: Extensible Markup Language (XML) 1.1 (second edition) W3C recommendation. (2006)

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. Scientific American. pp. 34–43. (2001)

[3] J. Kahan, M. Koivunen, E. Prud'Hommeaux, R. R. Swick. Annotea: An Open RDF Infrastructure for Shared Web Annotations, in Proc. of the WWW 10th International Conference, Hong Kong. (2001)

[4] J. P. Katoen: The principapl of Model Checking. University of Twente. (2002)

[5] K. Homma, K. Takahashi, A. Togashi. Modeling and Verification of Web Applications Using Formal Approach. IEICE Tech. Rep., vol. 109, no. 40, SS2009-8, pp. 43-48. (2009)

[6] R. Mateescu, S. Meriot, S. Rampacek. Extending SPARQL with Temporal logic. Technical report. (2009)

[7] E. D. Sciascio, M. F. Donini, M. Mongiello and G. Piscitelli, "Web Applications Design and Maintenance using Symbolic Model Checking,"in Proc. Seventh European Conference on Software Maintenance and Reengineering (CSMR '03), pp. 63-72. (2003)

[8] E. D. Sciascio, M. F. Donini, M. Mongiello and G. Piscitelli, "AnWeb: a System for Automatic Support to Web Application Verification," in Proc. International Conference on Software Engineering and Knowledge Engineering (SEKE '02), pp. 609-616. (2002)

[9] F. Ricca and P. Tonella, "Analysis and Testing of Web Applications, " in Proc. International Conference on Software Engineering (ICSE2001), pp. 25-34. (2001)

[10] F. Ricca and P. Tonella, "Web Site Analysis: Structure and Evolution, " in Proc. International Conference on Software Maintenance (ICSM2000), pp. 76-86. (2000)

[11] M. Han and C. Hofmeister, "Modeling and Verification of Adaptive Navigation in Web Applications, " in Proc. International Conference on Web Engineering (ICWE'06), pp. 329-336. (2006)

[12] M. Haydar, S. Boroday, A. Petrenko and H. Sahraoui, "Properties and Scopes in Web Model Checking," in Proc. IEEE/ACM International Conference on Automated Software Engineering (ASE '05), pp. 400-404. (2005)

[13] S. Yuen, K. Kato, D. Kato,and K. Agusa, "Web Automa A Behavioral Model of Web applications based on the MVC model, " Computer Software, vol. 22, no. 2, pp. 44-57. (2005)

[14] M. Haydar, A. Petrenko, and H. Sahraoui, "Formal Verification of Web Applications Modeled by Communicating Automata, " in Proc. International Conference on Formal Techniques for Networked and Distributed Systems (FORTE2004), LNCS 3235, pp. 115-132. (2004)

[15] J. P. Katoen. The principal of Model Checking. University of Twente. (2002)

[16] A. Pnueli. The temporal logic of programs. In proc. 18th IEEE Symp. Foundations of Computer Science (FOCS'77), Providence, RI, USA. pages 46-57. (1977)

[17] A. Cimatti, E. Clarke, F. Giunchiglia, M. Roveri. NμSMV: a new symbolic model checker. (2000)

[18] M. Ben-Ari. Principles of the SPIN Model Checker. Springer. ISBN: 978-1-84628-769-5. (2008)

[19] D. Becket, B. McBride: RDF/ XML Syntax Specification (Revised). W3C recommendation. (2004)

[20] T. Berners-Lee. W3C recommandation. (2007)

[21] V. Bönström, A. Hinze, H. Schweppe: Storing RDF as a graph. Latin American WWW conference, Santiago, Chile. (2003)

[22] R. E. Tarjan: Depth-First search and linear graph algorithm. SIAM Journal of Computing 1, 2, 146-160. (1972).

[23] B. Vatant. Metadata to describe resources (Semantic Web Languages). In Proceedings of the INRA Seminar : Metadata: changes and prospects. (2008)