RDF2NµSMV: Mapping Semantic Graphs to NµSMV Model Checker

Mahdi Gueffaz [*1], Sylvain Rampacek[*1], Christophe Nicolle[*1]

*LE2I UMR CNRS 5158, University of Bourgogne
BP 47870, 21078 Dijon cedex, France

[1] {Mahdi.Gueffaz, Sylvain.Rampacek, CNicolle}@U-bourgogne.fr

***Abstract*— The most frequently used language to represent the semantic graphs is the RDF (W3C standard for meta-modeling). The construction of semantic graphs is a source of numerous errors of interpretation. 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 between two areas: the Semantic Web and the model checking. For this, we developed a tool, RDF2NµSMV, which converts RDF graphs into NµSMV language. This conversion aims checking the semantic graphs with the model checker NµSMV in order to verify the consistency of the data. To illustrate our proposal we used RDF graphs derived from IFC files (Building Information Modeling). These files represent digital 3D building model. Our final goal is to check the consistency of the IFC files that are made from a cooperation of heterogeneous information sources (plumbers, architects, electricians, etc.)**

Keywords: ***Semantic graph, RDF, Model-checking, temporal logic, NµSMV, IFC, BIM.***

I. INTRODUCTION

The increasing development of networks and especially the internet has greatly developed the heterogeneous gap between information systems. In glancing over the studies about interoperability of heterogeneous information systems we discover that all works tend to the resolution of semantic heterogeneity problems. Now, the W3C (World Wide Web Consortium) suggests norms to represent the semantic by ontology. Ontology is becoming an inescapable support for information systems interoperability and particularly in the Semantic Web. Literature now generally agrees on the Gruber's terms to define an ontology: explicit specification of a shared conceptualization of a domain [1]. The physical structure of ontology is a combination of concepts, properties and relationships. This combination is also called a semantic graph.

Several languages have been developed in the context of Semantic Web and most of these languages use XML (eXtensible Markup Language) as syntax [2]. The OWL (Web Ontology Language) [3] and RDF (Resource Description Framework) [4] are the most important languages of the semantic Web, they are based on XML. OWL allows representing the ontology, and it offers large capacity machines performing web content. RDF enhances the ease of automatic processing of Web resources. The RDF (Resource Description Framework) is the first W3C standard for enriching resources on the web with detailed descriptions. The descriptions may be characteristics of resources, such as author or content of a website. These descriptions are metadata. Enriching the Web with metadata allows the development of so-called Semantic Web [5]. The RDF is also used to represent semantic graph corresponding to a specific knowledge modeling. For example in the AEC (Architecture Engineering Construction) projects, some papers used RDF to model knowledge from heterogeneous sources (electricians, plumbers, architects, etc.). In this domain, some models are developed providing a common syntax to represent building objects. The most recent is the IFC (Industrial Foundation Classes) [6] model developed by the International Alliance of Interoperability. The IFC model is a new type of BIM (Building Information Model) and requires tools to check the consistency of the heterogeneous data and the impact of the addition of new objects into the building.

As the IFC graphs have a large size, their checking, handling and inspections are a very delicate task. In [7] we have presented a conversion from IFC to RDF. In this paper, we propose a new way using formal verification, which consists in the transformation of semantic graphs into a model and verifying them with a model checker. We developed a tool called "RDF2NµSMV" that transforms semantic graphs into a model represented in NµSMV [8] language. After this transformation, NµSMV verifies the correctness of the model written in NµSMV language with temporal logic in order to verify the consistency of the data described in the model of the huge semantic graphs.

The rest of this paper is organized as follows. In Section 2 we present an overview of the semantic graphs, especially the structure of the RDF graphs and the model checking. Then, in Section 3, we describe the mapping of the semantic graphs into models and our approach is defined in Section 4. Finally, we end with the conclusion.

II. AN OVERVIEW OF SEMANTIC GRAPHS AND MODEL CHECKING

The RDF is also used to represent semantic graphs corresponding to a specific knowledge modeling. It is a language developed by the W3C to bring a semantic layer to the Web [9]. It allows the connection of the Web resources using directed labeled edges. The structure of the RDF documents is a complex directed labeled graph. An RDF document is a set of triples <subject, predicate, object> as shown in the Figure 1. 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 the "resource" types. A resource is identified by an URI (Uniform Resource Identifier) [10, 11].

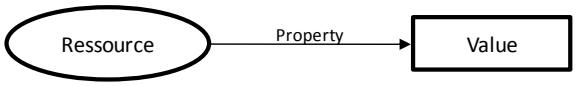
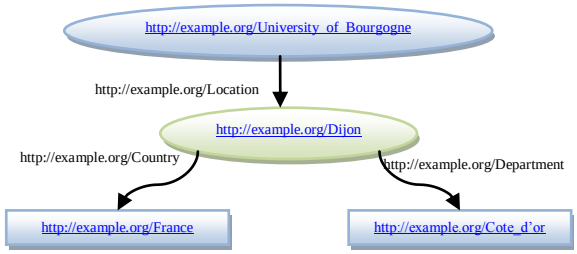


Figure 1. RDF triplet.

The declarations can also be represented as a graph, the nodes as resources and values, and the arcs as properties. The resources are represented in the graph by circles; the properties are represented by directed arcs and the 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.



Figure

2. Example of a partial RDF graph.

The RDF graph in the fig. 2 defines a node "University of Bourgogne" located at "Dijon", having as country "France" and as department "Cote d'Or". RDF documents can be written in various syntaxes, e.g., N3 [12], N-Triple [13], and RDF/XML. Below, we present the RDF\XML document corresponding to Figure 2.

```
<rdf:Description
rdf:about="http://example.org/university of
Bourgogne">
  <ex:Location>
      <rdf:Description
rdf:about="http://example.org/Dijon">
      <ex:Country> France</ex:Country>
      <ex:Department>Cote
d'or</ex:Department>
      </rdf:Description>
   </ex:Location>
</rdf:Description>
```

The model checking [14] described in fig. 3 is a verification technique that explores all possible system states in a brute-force manner. Similar to a computer chess program that checks all possible moves, a model checker, the software tool that performs the model checking, examines all possible system scenarios in a systematic manner. In this way, it can be shown that a given system model truly satisfies a certain property. Even the subtle errors that remain undiscovered using emulation, testing and simulation can potentially be revealed using model checking.

To make a rigorous verification possible, properties should be described in a precise unambiguous way. It is the temporal logic that is used in order to express these properties. The temporal logic is a form of modal logic that is appropriate to specify relevant properties of the systems. It is basically an extension of traditional propositional logic with operators that refer to the behavior of systems over time.

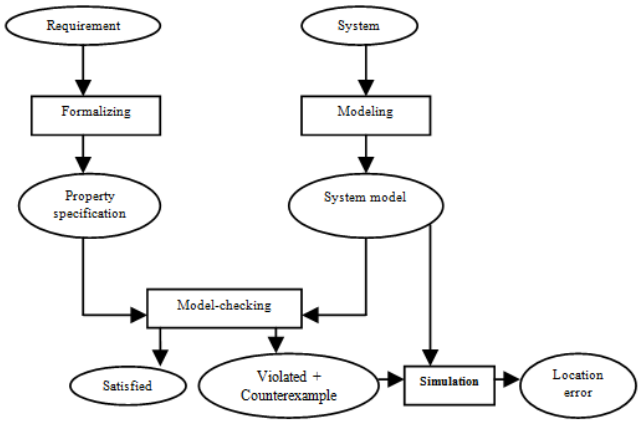


Figure 3. Model Checking approach

The following algorithm explains the way that the model checking works. First we put in the stack all the properties expressed in the temporal logic. All of them are verified one by one in the model and if a property does not satisfy the model, it is whether the model or the property that we must refine. In case of a memory overflow, the model must be reduced. Whereas formal verification techniques such as simulation and model checking are based on model description from which all possible system states can be generated, the test, that is a type of verification technique, is even applicable in cases where it is hard or even impossible to obtain a system model.

```
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
```

III. THE MAPPING

The RDF graphs considered here are represented as XML verbose files, in which the information is not hierarchically stored (so-called graph point of view). These RDF graphs 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.

A. Exploring RDF graph

In order to exploit the RDF graphs, we therefore have to determine whether they have a root vertex, 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 $E \subseteq VxV$ is the set of edges. For a vertex x, we note $E(x) = \{y \in V | (x, y) \in E\}$ the set of its successor vertices, and we assume that these vertices are ordered from $E(x)_0$ to $E(x)_{|E(x)|-1}$. 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:

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

We considered here an iterative variant of DFS, which makes use of an explicit stack, rather than the recursive variant given in [15]; this is required in practice to avoid overflows of the system call stack when the algorithm is invoked for exploring large graphs.

B. 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. 4), 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.

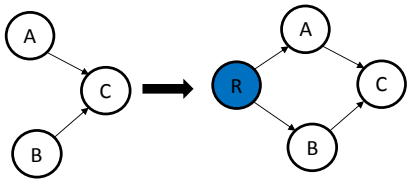


Figure 4. A root is a single node that has no predecessor. In this graph, we have node A and node B, two roots, and then we will create a new virtual root (blue circle "R") that points to the two roots.

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.

C. Generating the model

The third step is divided into three sub-steps. The first one consists in creating the table of all triplets by exploring the entire graph; the second one consists in generating the correspondence table and the last one in producing the model representing the semantic graph in NµSMV language.

Table of triplets – Going through the RDF graph by graph traversal algorithms, we will create a table consisting of resources, properties and values. In our RDF graph, the resource is a vertex, the property represents the edge and the value is the successor vertex corresponding to the edge of the vertex. The table of triplets of the RDF graph is useful to the next sub-step.

Correspondence table – In this second sub-step, RDF2NµSMV generates a table of correspondence. This table contains an identifier for each resource, property and value.

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

IV. The Verification with Model Checker

As we saw in Section 2, the model checker needs properties in order to check the model of semantic graphs. These properties are expressed in temporal logic. 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 analysed. 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.

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 a tool called "RDF2NµSMV" that transforms semantic graphs into NµSMV language.

We use NµSMV as model checkers to verify the model of semantic graphs. NµSMV is the amelioration of SMV model checker, working on the same simple principles as SMV. NµSMV verifies the properties in both linear time logic and computation tree logic.

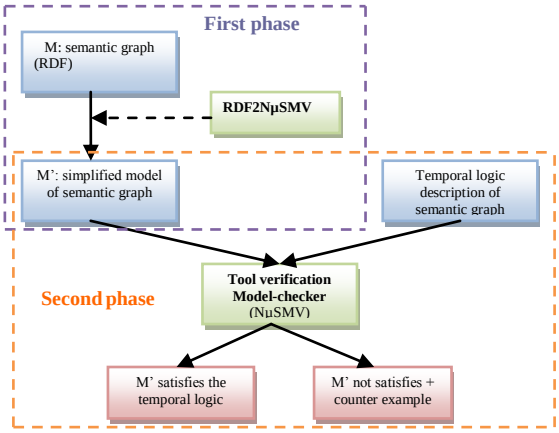


Figure 5. Our architecture.

The architecture of the fig. 5 is divided into two phases. The first phase concerns the transformation of the semantic graph into a model using our tool "RDF2NµSMV", as described in Section III. The second phase concerns the verification of the properties expressed in temporal logic on the model using the model-checker NµSMV.

To illustrate our approach, we take an RDF graph represented in the Figure 6 and a temporal logic expressed in the table 1 to verify if the BIM "b1" contains a floor.

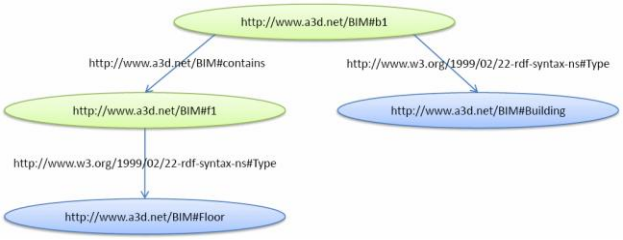


Figure 6. Example of partial RDF graph.

TABLE 1. Temporal logic formula.

Temporal logic	Meaning	Result
Eventually (b1 → Next Next floor)	Is there a floor after two states starting from the state b1	True

We tested several RDF graphs on our tool "RDF2NµSMV", graphs representing buildings as shown in Figure 7, using a machine that runs on a processor with a capacity of 2.4 GHz and 4 GB of RAM, calculating the time of conversion as shown in Fig 8. Note that the RDF2NµSMV tool is faster in converting semantic graphs. We have almost 22 seconds for a graph of 53 MB size. The transformation tool follows a polynomial curve. In Fig. 9, we see the size of the converted semantic graphs from RDF to NµSMV language.

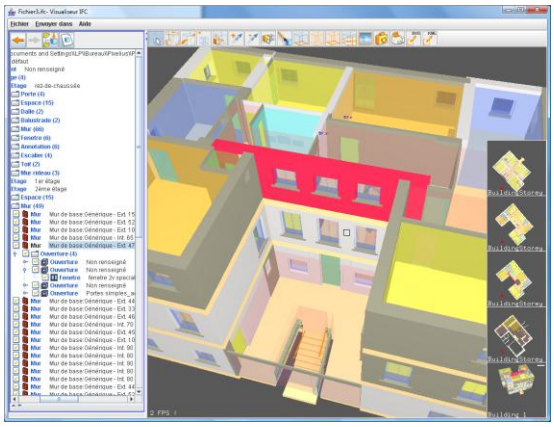


Figure 7. The 3D view of an IFC file.

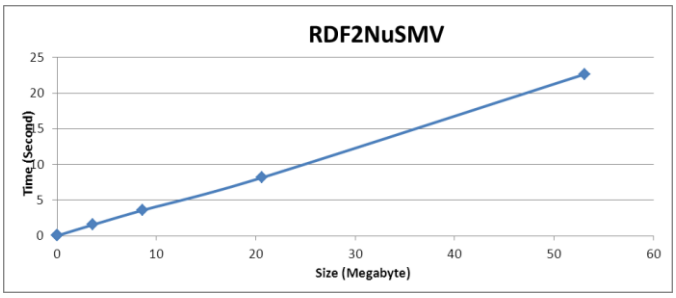


Figure 8. Time conversion of semantic graphs.

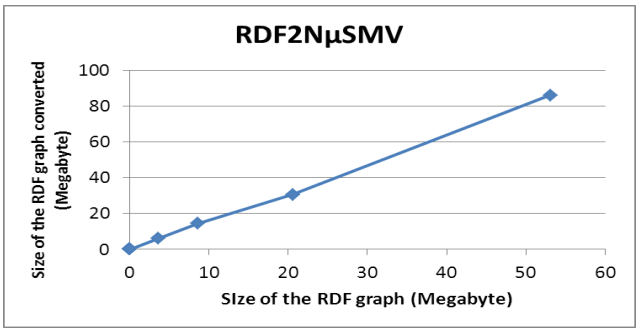


Figure 9. Size of the models

V. CONCLUSION

This paper presents how to transform a semantic graph into a model for verification by using a powerful formal method, that is the "model checking". Knowing that the model-checker does not understand the semantic graphs, we developed a tool RDF2NµSMV to convert them into NµSMV language in order to be verified with the temporal logics. This transformation is made for the purpose of classifying large semantic graphs in order to verify the consistency of IFC files representing 3D building. We notice the advantage of NµSMV, whose verification can be made with both linear time logic and computation tree logic formulas.

REFERENCES

[1] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing. Presented at the Padua workshop on Formal Ontology", March 1993, later published in International Journal of Human-Computer Studies, Vol. 43, Issues 4-5, November 1995, pp. 907-928.

[2] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan, "Extensible Markup Language (XML) 1.1 (second edition) W3C recommendation", http://www.w3.org/TR/2006/REC-xml11-20060816/. (2006)

[3] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneijder, and L. Andrea Stein, "OWL Web Ontology Language Reference", World Wide Web Consortium (W3C), http://www.w3.org/TR/owl-ref/, (2004).

[4] D. Becket and B. McBride, "RDF/ XML Syntax Specification (Revised)". W3C recommendation. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. (2004)

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

[6] IFC Model, Industrial Foundation classes, International Alliance for interoperability, http://www.buildingsmart.com/ (2008)

[7] R. Vanland, C. Nicolle, and C. Cruz, "IFC and Buildings Lifecycle Management", Journal of Automation in Construction, Elsevier, pp. 70-78 (2008).

[8] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: a new symbolic model checker". International Journal on Software Tools for Technology Transfer, pp. 495- 499. (2000)

[9] J. J. C. G. Klyne, "Resource Description Framework (rdf): Concepts and abstract syntax". Tech. rep., W3C. (2004)

[10] V. Bönström, A. Hinze, and H. Schweppe, "Storing RDF as a graph". Latin American WWW conference, Santiago, Chile, pp. 27-36. (2003)

[11] T. Berners-Lee, "W3C recommandation. http://www.w3.org/DesignIssues/ HTTP-URI". (2007)

[12] T. Berners-Lee and D. Connolly, "Notation3 (N3): A readable RDF syntax". W3C recommendation, http://www.w3.org/TeamSubmission/n3/. (2008)

[13] D. Becket and B. McBride, "RDF test cases. W3C Working draft". http://www.w3.org/TR/rdf-testcases/ (2004)

[14] J. P. Katoen, "Principles of Model Checking". Formal Methods and Tools Group. University of Twente, Lecture Notes, pp. 13-34. (2004)

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

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