International Journal of Future Generation Communication and Networking
Vol. 3, No. 1, March, 2010

Pure Petri Nets for Software Verification and Validation of Semantic Web Services in Graphical Worlds

Andres Iglesias

Department of Applied Mathematics and Computational Sciences, University of Cantabria, Avda. de los Castros, s/n, E-39005, Santander, SPAIN
iglesias@unican.es
http://personales.unican.es/iglesias

Abstract

Software verification and validation (SVV) are major ingredients of current software engineering projects. Among the available methods to solve this problem, formal methods (although very costly at the computational level) are arguably the most powerful ones. One of the most promising approaches in this field is that based on Petri nets. This paper discusses some issues regarding the application of Petri nets to SVV from a hybrid mathematical/computational point of view. The paper also describes a Mathematica package developed by the author for a class of Petri nets, which is applied to address the SVV problem in the context of graphical semantic web services based on virtual agents evolving in digital 3D worlds.

Keywords: *Software verification, software validation, Petri nets, semantic web services.*

1. Introduction

Nowadays, software verification and software validation are seen as integral parts of any software engineering project. Although there is a common misconception portrayed in many books and media (even in scientific papers) about both terms being synonymous, they actually refer to two different (and very often complementary) processes. *Software verification* tries to ensure that your final software matches the original design, i.e. you built your software according to the prescribed specifications. Such specifications should provide a complete description of the behavior of the system to be developed, including a library of use cases that describe all possible interactions between end users and the software. By contrast, *software validation* concerns the problem of checking whether your software satisfies or fits the intended usage, i.e. if your software is actually doing what the user really asks for.

Many different methods can be used to accomplish the previous tasks. They can be roughly classified as formal and syntactic methods. The former ones are focused on the operational and the axiomatic semantics, in which the meaning of the system is expressed in terms of preconditions and postconditions which are true before and after the system performs a task, respectively. In general, formal methods are mathematically-based techniques for proof correctness (including abstract state machines, process calculi, model checking and theorem provers, for instance) offering a high level of reliability but are often very costly and hence, only used in those fields where the benefits of having such proofs, or the danger in having undetected errors, makes them worth the resources. The syntactic methods are aimed at

looking for code failures by examining the structure of the code at its syntactic rather than its semantic level.

Among the myriad of formal methods for software verification and validation, those based on Petri nets (PN) are gaining more and more popularity during the last few years. Most of PN interest lies on their ability to represent a number of events and states in a distributed, parallel, nondeterministic or stochastic system and to simulate accurately processes such as concurrency, sequentiality or asynchronous control [4,16]. In addition, the mathematical foundations of Petri nets have been largely analyzed by means of many powerful techniques: linear algebraic techniques to verify properties such as place invariants, transition invariants and reachability; graph analysis and state equations to analyze their dynamic behavior; simulation and Markov-chain analysis for performance evaluation, etc. As a consequence, Petri nets provide the users with a very powerful formalism for describing and analyzing a broad variety of information processing systems both from the graphical and the mathematical viewpoints. Since its inception in the early 60s, they have been successfully applied to many interesting problems including finite-state machines, concurrent systems, multiprocessors, parallel and distributed computation, formal languages, communication protocols and many others.

In this paper we consider the use of Petri nets for software verification and validation (SVV). The paper also describes a *Mathematica* package developed by the author for a class of Petri nets, which is subsequently applied to address the SVV issue in the context of graphical web services based on agents. In particular, we analyze a case study dealing with a recently introduced web service framework [7]. Such framework is aimed at providing the users with web services by means of virtual agents resembling human beings and evolving within a 3D virtual world (a virtual representation of a real environment such as a shopping center or similar). Once a web service is requested on the client side via a web browser, the user is prompted into this virtual world (that is actually a replica of the real environment associated with the service) and immediately assigned his/her own virtual agent; in other words, the user is echoed by his/her virtual counterpart. The interplay between the users and the system is accomplished via those virtual agents, which are represented graphically in this virtual world and behave in a human-like way. More details about this system will be given in Section 5.

The structure of this paper is as follows: firstly, some basic concepts and definitions about Petri nets (mainly intended for those unfamiliar with this kind of methodology) are given in Section 2. Section 3 discusses the potential application of Petri nets to software verification and validation, while a *Mathematica* package (developed by the author) for dealing with some kinds of Petri nets is briefly reported in Section 4. The package is subsequently applied in Section 5 for software verification and validation of a recently introduced framework for web services. Some conclusions and further remarks close the paper.

2. Basic concepts and definitions

A *Petri net* (PN) is a special kind of directed graph, together with an initial state called the initial marking. The graph of a PN is a bipartite graph containing *places* $\mathbf{P}=\{P_1,\dots,P_m\}$ and *transitions* $\mathbf{T}=\{t_1,\dots,t_n\}$. Figure 1 shows an example of a Petri net comprised of three places and six transitions. In graphical representation, places are usually displayed as circles while transitions appear as rectangular boxes. The graph also contains arcs either from a place P_i to a transition t_j (*input arcs* for t_j) or from a transition to a place (*output arcs* for t_j). These arcs are labeled with their weights (positive integers), with the meaning that an arc of weight w can be understood as a set of w parallel arcs of unity weight (whose labels are usually

omitted). In Fig. 1 the input arcs from P_1 to t_3 and P_2 to t_4 and the output arc from t_1 to P_1 have weight 2, the rest having unity weight.

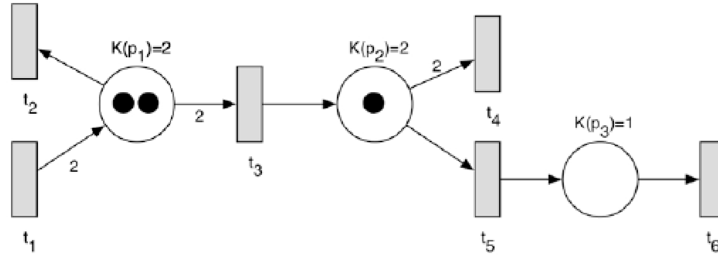

Figure 1. Example of a Petri net of three places and six transitions

A *marking* (state) assigns to each place P_i a nonnegative integer, k_i. In this case, we say that P_i *is marked with* k_i *tokens*. Graphically, this idea is represented by k_i small black circles (tokens) in place P_i. In other words, places hold tokens to represent predicates about the world state or internal state. The presence or absence of a token in a place can indicate whether a condition associated with this place is true or false, for instance. For a place representing the availability of resources, the number of tokens in this place indicates the number of available resources. At any given time instance, the distribution of tokens on places, called Petri net marking, defines the current state of the modeled system. All markings are denoted by vectors M of length m (the total number of places in the net) such that the i-th component of M indicates the number of tokens in place P_i. From now on the initial marking will be denoted as M_0. For instance, the initial marking (state) for the net in Figure 1 is *{2,1,0}*.

The pre-set and post-set of nodes are specified in this paper by a dot notation, where $\sum u=\{v \in \mathbf{P}\cup\mathbf{T} / (v,u) \in\mathbf{A}\}$ is called the *pre-set* of u, and $u\sum =\{v \in \mathbf{P}\cup\mathbf{T} / (v,u) \in\mathbf{A}\}$ is called the *post-set* of u (where **A** represents the set of arcs of the Petri net). The pre-set of a place (transition) is the set of input transitions (places). The post-set of a place (transition) is the set of output transitions (places). The dynamical behavior of many systems can be expressed in terms of the system states of their Petri net. Such states are adequately described by the changes of markings of a PN according to a *firing rule* for the transitions: a transition t_j is said to be *enabled* in a marking M when all places in $\sum t_j$ are marked. For instance, transitions t_2, t_3 and t_5 in Figure 1 are enabled, while transitions t_4 and t_6 are not. Note, for example, that transition t_4 has weight 2 while place P_2 has only 1 token, so arc from P_2 to t_4 is disabled. If transition t_j is enabled, it may or may not be fired (depending on whether or not the event represented by such a transition occurs). A firing of transition t_j removes $w_{i,j}$ tokens from each input place P_i of t_j and adds $w_{j,k}$ tokens to each output place P_k of t_j, $w_{j,k}$ being the weight of the arc from t_j to P_k. In other words, if transition t_j is fired, all places of $\sum t_j$ have their input tokens removed and a new set of tokens is deposited in the places of $t_j\sum$ according to the weights of the arcs connecting those places and t_j. For instance, transition t_3 removes two tokens from place P_1 and adds one token to place P_2, thus changing the previous marking of the net. The fireability property of a transition t_j is denoted by $M[t_j>$ while the creation of a new marking M' from M by firing t_j is denoted by $M[t_j>M'$.

A marking M^* is reachable from any arbitrary marking M *iff* there exists a sequence of transitions $S=t_1\,t_2\,t_3\ldots t_n$ such that $M[t_1>M_1[t_2>M_2\ldots M_{n-1}[t_n>M^*$. For short, we denote that the marking M^* is reachable from M by $M[S>M^*$, where S is called the *firing sequence*. The set of all markings reachable from M for a Petri net PN is denoted by $\approx[(PN,M)>$. Given a Petri

net *PN*, an initial marking M_0 and any other marking *M*, the problem of determining whether $M \in \approx[(PN,M_0)>$ is known as the *reachability problem* for Petri nets. It has been shown that this problem is decidable [10] but it is also *EXP-time* and *EXP-space hard* in the general case [11]. In many practical applications it is interesting to know not only if a marking is reachable, but also what are the corresponding firing sequences leading to this marking. This can be done by using the so-called *reachability graph*, a graph consisting of the set of nodes of the original Petri net and a set of arcs connecting markings M_i and M_j *if and only if* there exists $t \in \mathbf{T} / M_i[t>M_j$.

A transition without any input place is called a *source transition*. Note that source transitions are always enabled. In Figure 1 there is only one source transition, namely t_1. A transition without any output place is called a *sink transition*. The reader will notice that the firing of a sink transition removes tokens but does not generate new tokens in the net. Sink transitions in Figure 1 are t_2, t_4 and t_6. A couple (P_i,t_j) is said to be a *self-loop* if $P_i \in (\sum t_j \cap t_j \sum)$ (i.e., if P_i is both an input and an output place for transition t_j). A Petri net free of self-loops is called a *pure net*. In this paper, we will restrict exclusively to pure nets.

Some PN do not put any restriction on the number of tokens each place can hold. Such nets are usually referred to as *infinite capacity net*. However, in most practical cases it is more reasonable to consider an upper limit to the number of tokens for a given place. That number is called the *capacity* of the place. If all places of a net have finite capacity, the net itself is referred to as a *finite capacity net*. All nets in this paper will belong to this later category. For instance, the net in Figure 1 is a finite capacity net, with capacities 2, 2 and 1 for places P_1, P_2 and P_3, respectively. If so, there is another condition to be fulfilled for any transition t_j to be enabled: the number of tokens at each output place of t_j must not exceed its capacity after firing t_j. For instance, transition t_1 in Figure 1 is initially disabled because place P_1 has already two tokens. If transitions t_2 and/or t_3 are applied more than once, the two tokens of place P_1 will be removed, so t_1 becomes enabled. Note also that transition t_3 cannot be fired initially more than once, as capacity of P_2 is 2.

3. Petri nets for software verification and validation

Petri nets have been widely used as a formal method for software verification and validation during the last two decades. The reason is the large amount of mathematical tools available to analyse standard Petri nets. Indeed, a PN model can be described by a set of linear algebraic equations [16], or other mathematical models reflecting the behavior of the system [2]. This allows us to perform a formal check of the properties related to the behavior of the underlying system, e.g., precedence relations amongst events, concurrent operations, appropriate synchronization, freedom from deadlock, repetitive activities, and mutual exclusion of shared resources, to mention just a few. The simulation-based validation can only produce a limited set of states of the modeled system, and thus can only show presence (but not absence) of errors in the model, and its underlying requirements specification. The ability of Petri nets to verify the model formally is especially important for realtime safety-critical systems (such as air-traffic control systems, rail-traffic control systems, nuclear reactor control systems, etc.) and online operations (this is exactly the case of the example described in Section 5). On the other hand, they provide a powerful formalism for axiomatic semantics, so PN are very well suited for semantic web and related fields.

Based on these considerations, some PN-based models for software verification and validation have been developed. As a general rule, we can start by creating a reduced grammar reflecting only those context-free aspects of the language under consideration controlling the modelling [5]. As a consequence, this approach can readily be applied to any

given language. Then, we construct the PN components for each basis structure of the reduced grammar; for doing so, we basically follow the approach described in [5]. However, the description of the reduced grammar and their associated components is beyond the scope of this paper and will not be given here.

4. Mathematica package for Petri nets

In this section a *Mathematica* package (developed by the author) for dealing with Petri nets is described. For the sake of clarity, the main commands of the package will be described by means of its application to some Petri net examples. In this section we will restrict to the case of pure and finite capacity nets. We firstly load the package:

```
In[1]:= <<PetriNets`
```

A Petri net (like that in Figure 1 and denoted onwards as `net1`) is described as a collection of lists. In our representation, `net1` consists of three elements: a list of couples *{place, capacity}*, a list of transitions and a list of arcs from places to transitions along with its weights:

```
In[2]:=net1={{{p1,2},{p2,2},{p3,1}},{t1,t2,t3,t4,t5,t6},{{p1,
t1,2},{p1,t2,-1},{p1,t3,-2},{p2,t3,1},{p2,t4,-2},{p2,t5,-1},
{p3,t5,1},{p3,t6,-1}}};
```

Note that the arcs are represented by triplets *{place,transition,weight}*, where positive value for the weights mean output arcs and negative values denote input arcs. This notation is consistent with the fact that output arcs add tokens to the places while input arcs remove them. Now, given the initial marking *{2,1,0}* and any transition, the `FireTransition` command returns the new marking obtained by firing such a transition:

```
In[3]:= FireTransition[net1,{2,1,0},t2];
```
Out[3]:= {1,1,0}

Given a net and its initial marking, an interesting question is to determine whether or not a transition can be fired. The `EnabledTransitions` command returns the list of all enabled transitions for the given input:

```
In[4]:= EnabledTransitions[net1,{2,1,0}];
```
Out[4]:= {t2,t3,t5}

The `FireTransition` command allows us to compute the resulting markings obtained by applying these transitions onto the initial marking:

```
In[5]:= FireTransition[net1,{2,1,0},#]& /@ %;
```
Out[5]:= {{1,1,0},{0,2,0},{2,0,1}}

Note that, since transition t_1 cannot be fired, an error message is returned:

```
In[6]:= FireTransition[net1,{2,1,0},t1];
```
Out[6]:= FireTransition: Disabled transition: t1 cannot be fired for the given net and the {2,1,0} marking.

From *Out[4]* and *Out[5]*, the reader can easily realize that successive applications of the `EnabledTransitions` and `FireTransition` commands allows us to obtain all possible markings and all possible firings at each marking. However, this is a tedious and time-consuming task to be done by hand. Usually, such markings and firings are graphically

displayed in the reachability graph (see description above). Next input returns the reachability graph for our Petri net and its initial marking:

```
In[7]:= ReachabilityGraph[net1,{2,1,0}];
```
Out[7]:= See Figure 2

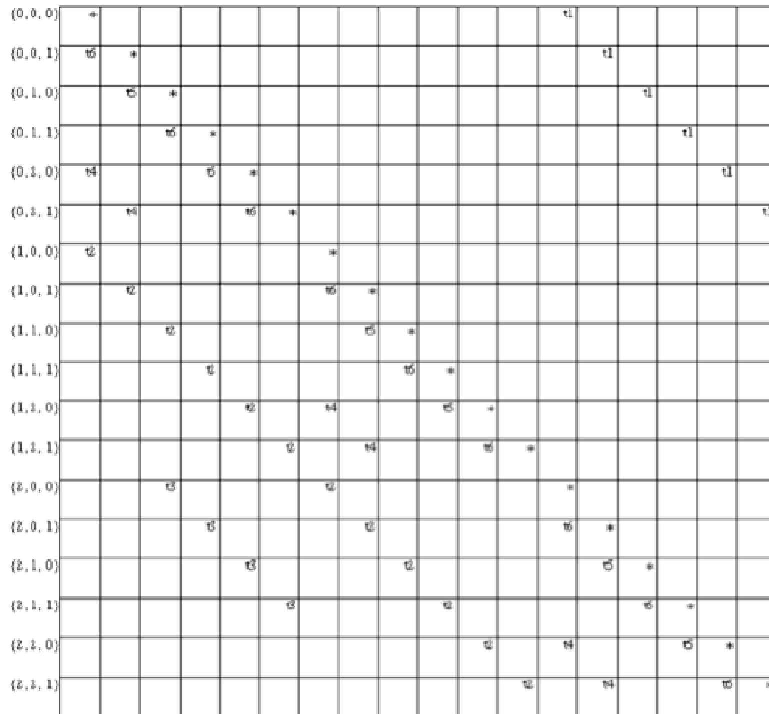

Figure 2. Reachability graph for *net1* with the initial marking {2, 1, 0}

Figure 2 can be interpreted as follows: the outer column on the left provides the list of all possible markings for the net. Their components are sorted in increasing order from the top to the bottom, according to the standard lexicographic order. For any marking, the row in front gives the collection of its enabled transitions. For instance, the enabled transitions for the initial marking *{2,1,0}* are *{t2,t3,t5}* (as expected from *Out[4]*), while they are *{t1,t4,t6}* for *{0,2,1}*. Given a marking and one of its enabled transitions, we can determine the output marking of firing such transition by simply moving up/down in the transition column until reaching the star symbol: the marking in that row is the desired output. By this simple procedure, results such as those in *Out[5]* can readily be obtained.

A second example of a Petri net is shown in Figure 3. This net, comprised of five places and six transitions, has many more arcs than the previous example. Consequently, its reachability graph, shown in Figure 4, is also larger. The *Mathematica* codes for defining the net and getting this graph are similar to those for the first example and therefore they are omitted for shortness.

The net in Figure 3 exhibits a number of remarkable features: for instance, places P_1, P_2 and P_5 have more than one output transition, leading to non-deterministic behavior. Such a structure is usually referred to as a *conflict*, *decision* or *choice*. On the other hand, this net has no source transitions. This fact is reflected in the reachability graph, which has a triangular

structure: entries appear only below the diagonal. As opposed to this case, the net in Figure 1 has one single source transition (namely, t_1), the only element above the diagonal in its reachability graph.

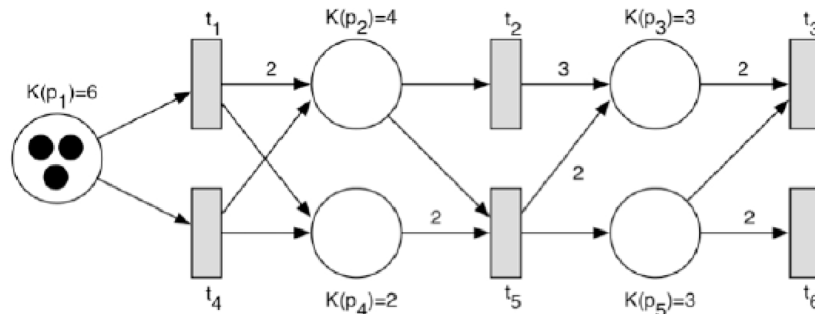

Figure 3. Petri net comprised of five places and six transitions

It is worthwhile to mention that the place P_1 has only input arcs, meaning that its number of initial tokens can only decrease, but never increase. This means that the capacity of P_1 might be less without affecting current results. On the other hand, the reachability graph in Figure 4 has some markings no further transitions can be applied onto. Examples of such markings are {1,3,3,2,0}, {1,2,3,2,0} or {0,4,3,1,0} (although not the only ones). They are sometimes called *end markings*.

5. Case study: graphical web services in virtual worlds

In a recent paper, the author described a new framework for semantic web services based on the so-called GAIVAs (Graphical Autonomous Intelligent Virtual Agents) [7]. The system was originally designed to fulfill a twofold objective: on one hand, it is a new approach to based-on-agents intelligent semantic web services: users can invoke web services interpreted by means of a sophisticated based-on-Artificial-Intelligence kernel. All "intelligent" tasks are performed by virtual agents that simulate human beings evolving within a virtual 3D world associated with the current web service. These agents are autonomous in the sense that they are able to take decisions and perform actions without human intervention. Of course, those decisions must be intelligent from the point of view of a human observer. On the other hand, the framework incorporates a powerful GUI (Graphical User Interface) that allows the users to interact with the system in a graphical and very natural way. Once a web service is requested, user is prompted through the web into a virtual world that is actually a replica of the real environment associated with the service [12,13]. The interplay between users and the system is accomplished via those virtual agents, which are represented graphically in this virtual world by their counterpart avatars who behave in a human-like way [14]. To the best of author's knowledge, no other approaches have considered this approach in the context of semantic web for services.

To show the performance of the proposal, we consider a simple yet illustrative example: a virtual shopping center, an environment that reproduces a real mall where users go to do shopping (see Figure 5 for two snapshots of this graphical 3D world). This scenario has been primarily chosen because it reflects one of the most typical services on the web - E-commerce - and provides the user with a bulk of potential agent-object and agent-agent interactions. The different shops in this virtual environment can easily be associated with real shops. To this

aim, the only programmer's needs are the basic information to be provided to the DAML-S tools for semantic web services, namely, what the services do, how they work and how they are used. This information is stored into a database of services. Pointers to this database allow the user to navigate through the different services associated with the shops, compare prices and items and carry out the most usual tasks of any similar real environment.

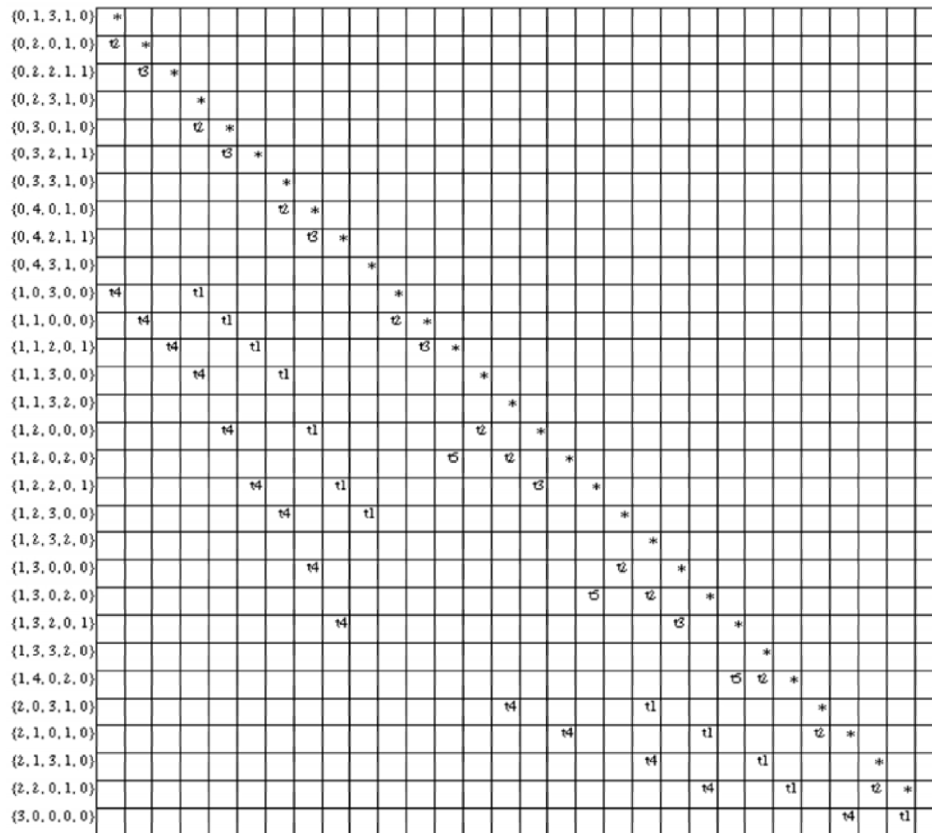

Figure 4. Reachability graph for the Petri net in Figure 3

The graphical tasks for the renderer system have been performed by using Open GL with GLUT (Open GL Utility Toolkit) for the higher-level functions (windowing, menus, or input) while Visual C++ v6.0 is used as the programming environment for better performance. To decrease bandwidth requirements, rough texture mapping for the virtual shops have been used when possible. A Prolog reasoner includes a based-on-rules expert system for making appropriate choices of items, based on the user requests and preferences. Those preferences are included as rules, and the inference engine performs the deductive processes [6]. The system asks the user about the services via the user interface. Some semantic tools (comprised of a DAML-S translator, a Knowledge Database and an OWL reasoner built in Prolog and not described here because of limitations of space) are then applied to interpret user's choices and proceed accordingly. The final output is returned to the user via a Web browser [15]. Users may do shopping from their home, office or anywhere else, and get all services currently available in real shopping centers at will. For instance, you can perform any banking service

(such as paying a bill, checking your balance account, or modifying your client's profile) by simply creating an agent that goes to the virtual bank office and asks for those services.

Figure 5. Two snapshots of the virtual shopping center

Security issues are performed in the usual manner through HTTPS secure pages and standard cryptographic procedures. The simulation framework can be implemented on a single CPU platform by creating a dynamic list of classes associated with the virtual agents. The communication between those classes and the behavioural system is achieved via DLLs to optimize the running speed. For virtual scenarios containing a large number of agents, objects and services, the alternative of distributed systems that associate each processor with a particular agent (or group of them) and communicating via threads leads to better performance ratios.

A critical problem in this framework is the verification and validation of the underlying software. The system has been designed for multi-task, multi-user on-line operations that typically require real-time processing and instantaneous access to the virtual world and their web services. Therefore, it is crucial to ensure that no thread/process deadlocks occur and that an adequate rendezvous of those threads/processes is achieved. Petri nets have proved to be very powerful tools to achieve such goals.

In order to make the problem affordable, some limitations on the number of simultaneous users and tasks are to be imposed. Such limitations come from two different scenarios: on the mathematical side, it is desirable to prevent the system from infinite-size (or finite but very huge) graphs. In addition, self-loops are not allowed and only finite capacity nets are considered. On the computational side, we need to set up an upper threshold for the number of connections and processes carried out at a given time, as the computational resources on the communicating parties (end users and service providers) can differ very much in terms of memory, processor, connection bandwidth and storage capacity. These constraints are also necessary because the current implementation is not the final product, but a trial prototype (with some bugs and improper declarations, as it will be explained later on). Of course, this upper threshold strongly relies on the computational architecture and available resources, which increase dramatically over the time. In the examples carried out so far up to 20 users and 100 simultaneous tasks have been considered.

Another important issue about verification and validation of our software is the fact that the system is comprised of many different modules working in a non-sequential way. Many

of those modules are not intended for web services but for other tasks (such as creating the graphical environment (the 3D world) and the graphical appearance of the virtual agents, establishing an access protocol and identity verification, etc.) Furthermore, each module is a collection of different processes, each consisting of one or several threads. While the threads share some common resources (such as memory space and addresses), the processes can be seen as independent actions at the hardware level. Consequently, the verification must be performed at the process layer rather than at the thread layer (which is much simpler but strongly dependent on other threads). This means that our basic structures must be combined to form "composite" structures to account for complex processes (see details below).

Under these constraints, the system is well represented by a finite pure Petri net, so the aforementioned techniques (as well as our *Mathematica* package) can be applied at full extent. Inspired by [17,18], the implementation described in this paper has been validated by interactive simulation. To this aim, we considered several hypothetical use cases for simulation and checked the corresponding results. Each use case provided us one or more scenarios that convey how the system should interact with end users or another system to achieve a specific business goal. We remark that these use cases neither describe the software internal flow, nor explain how that software will be implemented. They simply show the steps that user should follow in order to employ the software for doing his/her work.

As mentioned earlier, the evaluation process demands composite structures that are ultimately based on atomic structures. The DAML-S *composedOf* property specifies the control flow and data flow of its sub-structures. In this work we assumed the composite structures introduced in [17], namely *sequence*, *parallel*, *condition*, *choice*, and the some *iterate* classes (*loop*, *while*, *which*) of DAML-S as they have proved to be very efficient during the runtime evaluation process. For instance, a web service *F* can be reached under the *sequence* composite structure if there is a sequential composition of processes that achieves *F*; in other words, if there is an ocurrence sequence in the reachability graph yielding *F*. Similarly, *safety* is defined as lack of reachabilility to an unsafe state: a web service is safe if there is no occurrence sequence in the net reaching that unsafe marking. Further, a marking state of a net is a *deadlock* if it enables no transitions. The search of deadlocks can be described as the search of reachable markings in the net leading to deadlocks. These examples show that *the SVV problem can be adequately addressed by using the reachability graph of Petri nets*, provided that the issues under analysis are accurately described in terms of markings in the net. If so, *the problem merely reduces to a search of reachable markings in the reachability graph*. The *Mathematica* package described earlier is author's tool for doing this task.

Regarding our evaluation results, they were according to our expectations - the Web services we tested behave quite well - but some other limitations became evident during the process. The most important ones have been, on one hand, the security access and authentication and, on the other hand, the integrity and correctness of some data. However, they are two problems of a completely different nature: the first problem lies on the framework layer, and can be explained by the fact that no specific modules have been designed for this issue (in fact, this is part of our planned future work). On the contrary, the limitations about the integrity and correctness lie on the implementation layer - actually, they were mostly due to some improper declarations at different parts of the code, readily found through standard debugging procedures - meaning that they do not affect the general structure of our framework.

6. Conclusions and further remarks

This paper discusses some issues regarding the application of Petri nets to software verification and validation (SVV). The paper also describes a *Mathematica* package developed by the author for a class of Petri nets, which is applied to address SVV problem in the context of graphical semantic web services based on virtual agents.

In our approach, interactions between end users and the system are accomplished by means of virtual agents. These agents exhibit a human-like physical appearance in the virtual world, thus emphasizing the expressive power of our scheme. Indeed, it has been pointed out that graphical interfaces have a beneficial effect for all users, even for those who are already familiar with Web services. This statement has been supported by some recent papers that have analyzed the interface agents from users' viewpoint. For instance, [9] and [20] studied user's responses and reactions to interfaces with static and animated faces, concluding that users found them to be more engaging and entertaining than functionally equivalent interfaces without a face. In [1,3] authors found that users prefer interfaces with agents and rate them a more entertaining and helpful than an equivalent interface without the agent. Agent's physical appearance is also very important: in [8,19] the authors reported that users were more likely to be cooperative with an interface agent when it had a human face (as opposed to a dog image or anything else). All these studies give a clear indication that the inclusion of virtual agents having a human-like appearance and behaviour greatly improves the efficiency of the communication channel and encourages people to use the system. Our framework provides an effective and seamless way to include all those features on standard software tools for web services.

Another positive feature of our framework is the inclusion of autonomy for the virtual agents. Most graphical interfaces based on agents - such as the graphical chat systems - do use virtual agents, but those agents are not autonomous. Therefore, user is forced to switch between controlling agent's behaviour and carrying out other actions. While the user is busy with those actions, the virtual agent keeps motionless or repeats a sequence of prescribed movements. This kind of answer causes misleading and conflicts between what users expect from the system and what they really get. This is a very important - and not sufficiently analyzed yet - issue for web services. In our approach, autonomy is provided by the knowledge motor via a combination of different Artificial Intelligence techniques so that the agents are able to evolve freely without human intervention (see [6,7,14] for details on the behavioral engine).

As abovementioned, the current system is just a draft prototype, not a final software product. As such, lots of improvements (even further theoretical research) are still needed. The proposal discussed here is only a first step in this walk. Future works include the improvement of graphical rendering and computational efficiency of the system, the consideration of a large number of users and tasks, the debugging of the source code, the extension of the *Mathematica* package for other kinds of Petri nets and the formulation of the SVV by using the PN state equations at full extent.

This paper is an extended and improved version of a previous contribution invited for presentation at FGCN’2009 conference, held in beautiful Jeju island (Korea) in November 2009. The author would like to thank FGCN'2009 conference chairs, and specially Prof. Tai Hoon Kim, for their kind invitation to deliver a talk at that conference. This research has been supported by the Computer Science National Program of the Spanish Ministry of Education and Science, Project Ref. #TIN2006-13615. Partial financial support from the University of Cantabria is also kindly acknowledged.

References

[1] Andre, E., Rist, T., Muller, J.: "Integrating reactive and scripted behaviours in a life-like presentation agent". In Proceedings of AGENTS'98 (1998) pp. 261-268.

[2] Bourdeaud'huy, T., Hanafi, S., Yim, P.: "Mathematical programming approach to the Petri nets reachability problem". European Journal of Operational Research 177 (2007) pp. 176-197.

[3] Cassell, J., Pelachaud, C., Badler, N., Steedman, M., Achorn, B., Becket, T., Douville, B., Prevost, S., Stone, M.: Animated conversation: rule-based generation of facial expression, gesture and spoken intonation for multiple conversational agents. In: Proceedings of ACM SIGGRAPH '94 (1994) pp. 413-420.

[4] German, R.: Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets. John Wiley and Sons, Inc. New York (2000).

[5] Heiner, M.: Petri Net Based Software Validation, Prospects and Limitations. Technical Report TR92-022, GMD/First at Berlin Technical University, Germany (1992).

[6] Iglesias, A., Luengo, F.: New Goal Selection Scheme for Behavioral Animation of Intelligent Virtual Agents. IEICE Transactions on Information and Systems, E88-D(5) (2005) pp. 865-871.

[7] Iglesias, A.: A new framework for intelligent semantic web services based on GAIVAs. Int. Journal of Information Technology and Web Engineering, 3(4) (2007) pp. 30-58.

[8] Kiesler, S., Sproull, L.: Social human-computer interaction. In: Human Values and the Design of Computer Technology, 199, CSLI Publications, Stanford, CA. (1997).

[9] Koda, T., Maes, P.: Agents with faces: the effects of personification of agents. In: Proceedings of Fifth IEEE International Workshop on Robot and Human Communication (1996) pp. 189-194.

[10] Kosara ju, S.R.: Decidability of reachability in vector addition systems. In: Proc. 14th Annual ACM Symp. Theory Computing. (1982) pp. 267-281.

[11] Lipton, R.: The reachability problem requires exponential space. Technical report, Computer Science Department, Yale University (1976).

[12] Luengo, F., Iglesias, A.: A New Architecture for Simulating the Behavior of Virtual Agents. Lectures Notes in Computer Science, 2657 (2003) pp. 935-944.

[13] Luengo, F., Iglesias, A.: Framework for Simulating the Human Behavior for Intelligent Virtual Agents. Part I: Framework Architecture. Lectures Notes in Computer Science, 3039 (2004) pp. 229-236.

[14] Luengo, F., Iglesias, A.: Framework for Simulating the Human Behavior for Intelligent Virtual Agents. Part II: Behavioral System. Lectures Notes in Computer Science, 3039 (2004) pp. 237-244.

[15] Luengo, F., Contreras, M., Leal, A., Iglesias, A.: Interactive 3D Graphics Applications Embedded in Web Pages. Proc. Computer Graphics, Imaging and Visualization-CGIV'2007 - Bangkok (Thailand), IEEE CS Press, Los Alamitos, California (2007) pp. 434-440.

[16] Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE, 77(4) (1989) pp. 541-580.

[17] Narayanan, S., McIlraith, S.: Simulation, Verification and Automated Composition of Web Services. In Proceedings of the Eleventh International World Wide Web Conference-WWW2002, ACM Press (2002).

[18] Narayanan, S., McIlraith, S.: Analysis and Simulation of Web Services. Computer Networks. 42 (2003) pp. 675-693.

[19] Sproull, L., Subramani, R., Kiesler, S., Walker, J., Waters, K.: When the interface is a face. In: Proceedings of Human-Computer Interaction. 11 (1996) pp. 97-124.

[20] Takeuchi, A., Naito, T.: Situated facial displays: towards social interaction. Proceedings of CHI'95, ACM Press (1995) pp. 450-455.

Authors

ANDRES IGLESIAS is currently the head of the Department of Applied Mathematics and Computational Sciences of the University of Cantabria (Spain). Since Nov. 2005, he has also been the Post-graduate studies coordinator at his department, awarded with the Quality Certificate by the Spanish Ministry of Education and Science. He has been a Visiting Researcher at (among others) the Department of Computer Science of the University of Tsukuba (Japan), Wessex Institute of Technology (UK), International Center of Theoretical Physics-ICTP (Italy) and Toho University (Japan). He holds a B.Sc. degree in Mathematics (1992) and a Ph.D. in Applied Mathematics (1995). He has been the chairman and organizer of 30 international conferences in the fields of computer graphics, geometric modeling and symbolic computation, such as the CGGM (2002-09), TSCG (2003-08) and CASA (2003-10) annual conference series and co-chair of ICMS'2006, VRSAL'2008, ICCIT'2008 and CGVR (2009-10). In addition, he has served as a program committee and/or steering committee member of over 100 international conferences such as 3CM, 3IA, ACN, CGA, CAGDAG, CGI, CGIV, CIT, CyberWorlds, FGCN, GMAG, GMAI, GMVAG, Graphicon, GRAPP, ICCS, ICCSA, ICICS, ICCIT, ICM, ICMS, IMS, IRMA, ISVD, MMM, NDCAP, SEPA, SMM, VIP, WSCG and WTCS. He has been reviewer of 91 international conferences, 22 international journals (including 13 ISI-indexed journals) and outstanding research institutions and agencies such as NSF (USA) and the European Commission. He is currently the Editor in Chief of the "International Journal on Computer Graphics", Associate Editor of the journals "International Journal of Computer Graphics and CAD/CAM", "Transactions on Computational Science", "Advances in Computational Science and Technology", "International Journal of Computational Science", "International Journal of Biometrics", "Journal of Convergence Information Technology", "Int. Journal of Future Generation Communication and Networking" and "International Journal of Digital Content Technology and its Applications" and member of the Editorial Reviewing Board of the "International Journal of Information Technology and Web Engineering". He has also been guest editor of some special issues of international journals about computer graphics and symbolic computation. He is the author of over 100 international papers on different topics and 7 books. For more information, take a look at his personal web site available at: http://personales.unican.es/iglesias