Analysis and Verification of Web Services Resource Framework (WSRF) Specifications Using Timed Automata

José A. Mateo, Valentín Valero, Enrique Martínez and Gregorio Díaz
Department of Computer Science
University of Castilla-La Mancha
Albacete, Spain
{*jmateo, gregorio, valentin, emartinez*}*@dsi.uclm.es*

***Abstract*—Throughout the history of computing, engineers have used various formal methods to improve the quality of software and hardware. The next natural step is trying to exploit their advantages in the so-called new era of computing: Cloud Computing. In this paper, we present a first approximation about how to simulate and check the behaviour of these systems using timed automata through the model checking tool UPPAAL. We use Web Services Resource Framework (WSRF) as a standard intended to the modelling of distributed resources using Web services, and we apply formal techniques to WSRF specifications in order to analyse and verify these specifications.**

***Keywords*-Web Services Resource Framework (WSRF); model checking; timed automata.**

I. INTRODUCTION

The architecture that represents Web services has been widely accepted as a means of structuring the interactions between services in a distributed system. Nowadays, developers require more standardization to facilitate additional interoperability between these services. In January of 2004, several members of the organization *Globus Alliance* and the multinational company *IBM*, with the help of experts from companies such as *HP*, *SAP*, *Akamai*, *etc.*, defined the basic architecture and the initial specification documents of a new standard for that purpose [6]. Web services Resource Framework (WSRF) has been inspired by the work previously done by *Global Grid Forum's Open Grid Services Infrastructure (OGSI) Working Group* [12]. Although a Web service definition does not consider the notion of state, interfaces frequently provide the user with the ability to access and manipulate states, i.e., data values that persist across, and evolve as a result of Web service interactions. However, the notion of stateful resources defined by the Web service implementation is not explicit in the interface definition. The messages that the services send and receive imply (or encourage programmers to infer) the existence of an associated stateful resource type. It is then desirable to define Web service conventions to enable the discovery of, introspection on, and interaction with stateful resources in standard and interoperable ways [4]. These observations motivated the WSRF approach to model Web services resource states. A WS-Resource is defined as the composition of a Web service and a stateful resource. WSRF allows WS-Resources to be declared, created, accessed, monitored for change, and destroyed via conventional mechanisms. WSRF consists of a set of five technical specifications that define the normative description of the WS-Resource approach in terms of specific message exchanges and related XML definitions.

In this paper, we propose the use of formal techniques and, more specifically, timed automata as a way to analyse and verify WSRF specifications. Thus, formal methods are used to write specifications that show the behaviour of the systems in a formal manner, and serve as the basis for system analysis to search for inconsistencies or errors in an early state of the development process, that is, before implementation. Furthermore, within formal methods environment, we use model checking techniques. Model checking [3] is an automatic technique for verifying finite-state reactive systems. In this approach, the specifications are expressed in a propositional temporal logic, and the reactive system is modelled as a state-transition graph (automaton). An efficient search procedure is used to determine automatically if the specifications are satisfied by the automaton. Model checking has a number of advantages over verification techniques based on automated theorem proving. The most important is that the procedure is highly automatic so it makes the testing phase faster. Typically, the user provides a high level representation of the model and the specification to be checked. The model checker will either terminate with the true answer, indicating that the model satisfies the specification, or giving a counterexample that shows why the formula is not satisfied.

As far as we know the literature in this field, no one has modelled the communication model in WSRF. Nevertheless, there are some works that use WSRF in a practical way. In [10], the authors presented a meta-model of a medical system for deriving clinical trial information management systems for collaborative cancer research across multiple institutions. With this meta-model, they extract the corresponding semantics in the Z formal specification language and the WSRF implementation in the real environment CancerGrid. The main difference with our work is the different

 ISBN: 978-1-61208-124-3

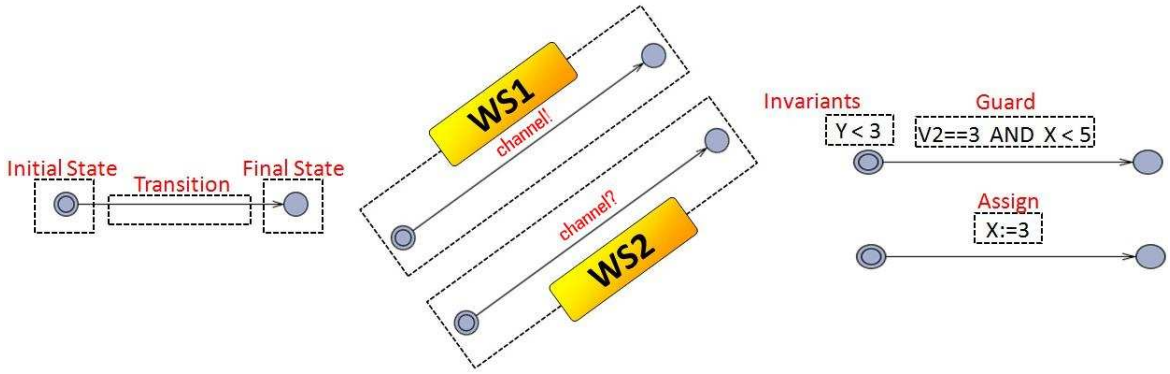

Figure 1. Examples of UPPAAL timed automata.

formalism they use to capture the behaviour of the system and no verification is done in this paper. Other related work is [7] where Gudelj et al. suggest a similar problem to ours. In this approach, they use a Petri net formalism to model the actions to be performed by the actors in this scenario, adding AI techniques (genetic algorithms) as another form of modelling. Indeed, they only show the possible prototype, not making verification. We can mention other related works: [11] and [8]. In [11], WSRF is used to solve the integration problem of various heterogeneous systems in a health information system grid model. In [8], the design and implementation of a Grid-based architecture for scientific workflow is presented. This architecture allows the dynamic discovery of existing Web services in combination to ad-hoc developed ones.

One of the main contributions of this paper is to define a primer version of the necessary elements to model and check Web services with stateful resources. The corresponding translation into timed automata will then be defined. In some previous works, such as [2] and [5], the verification of Web services compositions by means of timed automata has been considered, but without stateful resources. The other main contribution is to develop a scheduling meta-model with one part of the WSRF´s specifications (WS-Notification and WS-ResourceProperties) improving this work with a verification phase, using the UPPAAL model checker.

The rest of the paper is structured as follows: Section II contains the needed background of our approach, that is, the UPPAAL tool, and the WSRF specification. In Section III we specify the elements we need to model and check Web services with stateful resources, and the translation of these elements into timed automata. A case study is included to show how the approach works. Section IV shows how the verification process is carried out over the case study. Finally, Section V contains the conclusions and future work.

II. BACKGROUND

A. UPPAAL

UPPAAL [9] is a tool box for modelling, simulation, validation and verification of real-time systems, based on constraint-solving and on-the-fly techniques, developed jointly by the Uppsala University and the Aalborg University. It is appropriate for systems that can be modelled as a collection of non-deterministic processes with finite control structures and real-valued clocks, communicating through channels and (or) shared variables. Thus, a UPPAAL system consists of a set of concurrent processes, each of which being modelled by a timed automaton. This automaton consists of a set of nodes and a set of transitions. To define the behaviour of the system it is possible to define “invariants”, “guards” and “synchronizations” in the automata:

- The “synchronization” between processes is done through “channels”. One of the processes, which is called the initiator of the synchronization, will invoke the channel with the symbol “!”, while the other process will invoke the channel with the symbol “?”.
- A “guard” is a trigger condition of a transition. It expresses a condition over clocks and integer variables, which must be satisfied when the transition is taken.
- An “invariant” is a condition of progression associated with a node. It indicates the time that the automaton can remain in that node.

Figure 1 depicts some examples of timed automata representations in UPPAAL. On the left-hand side we can see how the states (nodes) of the automata are represented and the transitions between these states. On the centre we can see how two automata representing two different Web services (WS1 and WS2) can be synchronized by means of a channel. Finally, on the right-hand side we can see how invariants, guards and assignments are represented in the automata.

B. Web Services Resource Framework (WSRF)

WSRF [1] is a specification developed by OASIS (Organization for the Advancement of Structured Information

 ISBN: 978-1-61208-124-3

Name	Describes
WS-ResourceLifetime	Mechanisms for WS-Resource Destruction, including message exchanges that allow a requestor to destroy a WS-Resource.
WS-ResourceProperties	Definition of a WS-Resource, and mechanisms for retrieving, changing, and deleting WS-Resource properties.
WS-RenewableReferences	A conventional decoration of a WS-Addressing endpoint reference with policy information needed to retrieve an updated version of an endpoint reference when it becomes invalid.
WS-ServiceGroup	An interface to heterogeneous by-reference collections of Web services.
WS-BaseFaults	A base fault XML type for use when returning faults in a Web services message exchange.

Figure 2. WSRF technical specification.

Standards) and some of the most pioneering computer companies, whose purpose is to define a generic framework for modelling Web services with stateful resources, as well as the relationships between these services in a Grid/Cloud environment. This approach consists of a set of specifications that define a representation of WS-Resource in the terms that specify the messages exchanged and the related XML documents. These specifications allow the programmer to declare and implement the association between a service and one or more resources. It also includes mechanisms to describe the means to check the status of a resource and the service description, which together form the definition of a WS-Resource. Furthermore, they define the necessary steps to make the state of a Web service accessible through its interface (described in WSDL) and related mechanisms to addressing and grouping defined elements in the WS-Resource. WSRF is useful to declare, create, access, monitoring and destroying WS-Resources through conventional mechanisms. These conventional mechanisms are described as follows (Figure 2 summarizes some of them):

- *WS-ResourceLifetime*: The lifetime of a WS-Resource is defined as the period between its instantiation and destruction. The mission of this specification is to standardize the process of destroying a resource and identify mechanisms to monitor this lifetime, but this specification does not define how to create the WS-Resource. It includes two ways to destroy a resource: immediately through an explicit message or timed destruction.
- *WS-ResourceProperties*: WSRF uses a precise specification to define the properties of the WS-Resources. This definition will consist of the definition of the interface in WSDL and an XML document (Resource Properties Document) that specifies the properties of the associated resource, for example, the disk size, processor capacity, etc. If the user wants to access, modify or update this document it is necessary to use a series of messages defined by the specification.
- *WS-ServiceGroup*: This specification allows the creation of groups that share a common set of properties, i.e., it is a mechanism for grouping together different Web services with similar behaviour.
- *WS-Basefaults*: The developer typically uses a Web service interface defined by others, so a method to standardize the format for reporting error messages facilitates the work. This is the main goal of WS-BaseFaults.
- *WS-Notification*: This specification allows a *NotificationProducer* to send a notification message to a *NotificationConsumer* in two ways:
  1) The *NotificationProducer* sends a notification message to the *NotificationConsumer* without following any formalism.
  2) The *NotificationProducer* uses a specific formalism to send notifications.

  The option selected is sent by the subscriber in the subscription message. Thus, the second option allows the user to receive a wide range of notification messages, but the user can receive many topics in which they are not interested because the information sent in these messages is obtained from a topics tree stored in the Web service.
- *WS-BrokeredNotification*: A *NotificationBroker* is an intermediary, who, among other things, allows interactions between one or more *Publishers* and one or more *NotificationConsumers*. The mission of the *Publisher* is to observe situations and create notification messages to report these situations, while the broker is responsible for forwarding these messages.

III. Service + Resource Modelling

In this section, we show the necessary elements to model and check Web services with stateful resources and the corresponding translation into timed automata for verification.

Concerning the broker Web service, we need four channels to model the actions that this service can support: *Notification* (to send notifications to the others services), *QueryChannel* (to receive information), *ResponseChannel* (to send information), *PublishChannel* (to publish the information about the topics) and, finally, *SubscribeChannel* (to receive requests of subscription to one or more topics). In addition, we have three variables: *v* that represents the value of the data received or sent, *op* is the operation to perform and *id* is the identifier of the variable. These channels and variables have the same meaning in the Web service automaton. Figure 3 depicts the automaton for broker Web service.

Figure 4 depicts the automaton that models the Web service behaviour. The main difference between this automaton and the previous one is the task assigned to the channels

 ISBN: 978-1-61208-124-3

(receive or send data depending on the direction of the communication).

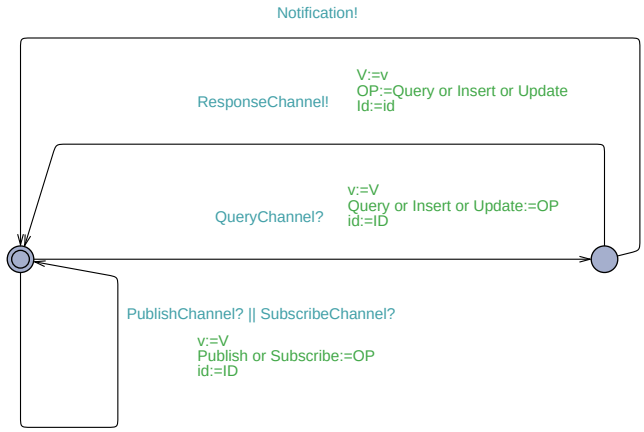

Figure 3. Broker service automaton.

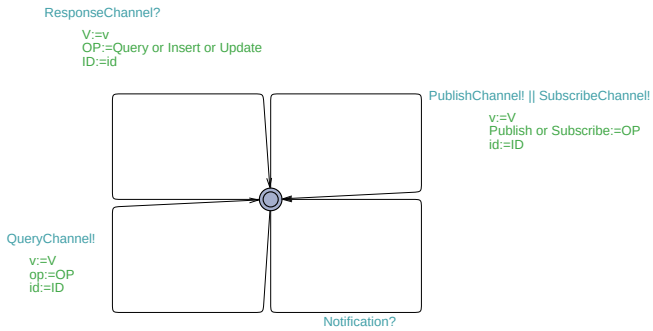

Figure 4. Web service automaton.

A. Case Study: CONTAINER TERMINAL PLANNING

Our modelling problem is a resource allocation for a series of particular tasks. In the case of WSRF, these tasks can be Web services and the resources would be the associated stateful resources. The description of the particular problem is the following: Given a number of trucks (tasks) with designated unloading window (in our case, the time between the WS-Resource creation and its timed destruction), assign the cranes (resources) to them, supposing that our system controls each one of the cranes at the port. When the trucks are near the port (10 Km.), they must report to the crane control tower (the broker role in WSRF) that they want to subscribe to the topic *CraneFree* so they can receive a notification of when they can unload their load. The broker assigns the cranes based on time windows, so it always chooses the truck with the smaller time window to ensure the system correct behaviour. Once the truck has finished its work, it must send a notification to the broker (*CraneFree*). For simplicity, we will call *Crane* the cranes control tower automaton and *Truck* the truck automaton. Figures 5 and 6 depict the cranes control tower and truck automata respectively.

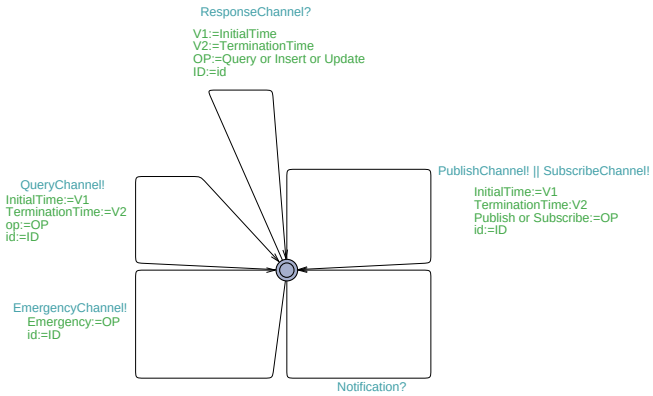

Figure 5. Control tower automaton.

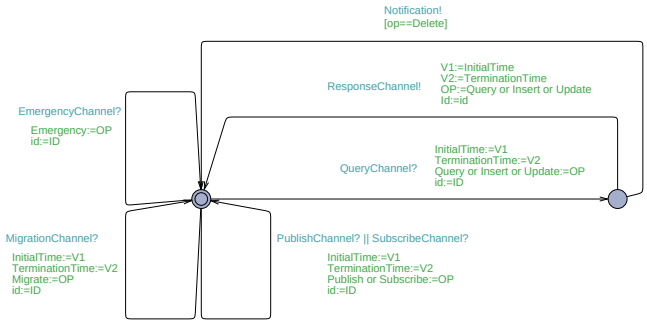

Figure 6. Truck automaton.

Next, we start by explaining the operation of the system in normal conditions and, after that, describing the exceptional behaviour. When a truck approaches to the port, it uses the *SubscribeChannel* to report its arrival time (*InitialTime*), its completion time (*TerminationTime*) and its identifier (*id*). The tower receives the information by its *SubscribeChannel* (so, we use ?) and immediately activates the *QueryChannel* to introduce this information in the database (op == Insert). The crane control tower responds with the information sent to it to find inconsistencies in the stored data. The trucks can query (op == query) and update (op == update) this data at any time by following the same steps.

 ISBN: 978-1-61208-124-3

In the event of receiving an emergency message, the system will insert the truck data at the top of the database (first position to unload) . Then, it sends a notification to the trucks which are approaching to the cranes for checking whether it is possible or not to unload later.

If it is a migration from one crane to another, the control tower will receive requests through *MigrationChannel*, proceeding as in the first case.

Finally, when a truck ends up its work, it notifies this situation to the tower by using *PublishChannel* and it proceeds to remove its information of the database (op==delete). Next, the control tower uses the channel *Notification* to give work permission to another truck.

IV. VERIFICATION

In the previous section, we have shown the translation between the communication model of the container terminal in WSRF and the corresponding timed automata. However, this work would be incomplete without a first approximation of how to model the task scheduling (trucks) in the terminal.

In this section, we present a simple model that represents the system in a general way and after that, we find very important to provide some formulas to check the correctness of the model. In this sense, as noted in the introduction, we use the UPPAAL model checker to ensure deadlock-freeness and search for possible errors to improve our system design. We model the internal behaviour of Web services, i.e., the necessary actions performed by the actors in this scenario to succeed in managing the scarce resources. Note that the figures of this section show a simplified model to ease understandability. The truck timed automaton has been modified to take into account two possible situations: **OnTime** or **Delayed**. Thus, if the unloading of goods is within the time window, the truck is considered on time, while on the other case, it is delayed. On the left-hand side of Figure 7 we show the representation of the *Crane* automaton. This timed automaton uses two channels: request and notification. The first one is used to accept service requests by trucks while the other channel is used to accept the notifications when the trucks end up their work. On the right-hand side, the *Truck* automaton is used to model the different states of the truck.

The next example will help the reader clarify the meaning of transitions and states. Assuming a certain time of arrival t_i, $mint_{early}$ defining the early time in which the truck can arrive to the port and $maxt_{late}$ representing the latter time in which the truck can leave the port, we need to define two possible situations: **OnTime** or **Delayed**. The first one is when the truck can arrive to the port between the interval $[mint_{early}, t_i]$ and the other one is when the truck arrives to the port in $[t_i + 1, maxt_{late}]$. The meaning of the channels is analogous to the *Crane* automaton. In Table I we show the trucks arrival timetable to the port.

TRUCKS	$Mint_{early}$	$Maxt_{early}$	$Mint_{late}$	$Maxt_{late}$
truck1	153	159	160	559
truck2	100	125	126	347
truck3	91	136	137	512
truck4	50	100	101	250
truck5	175	235	236	350
truck6	210	299	300	600

Table I
TRUCKS ARRIVAL TIMETABLE.

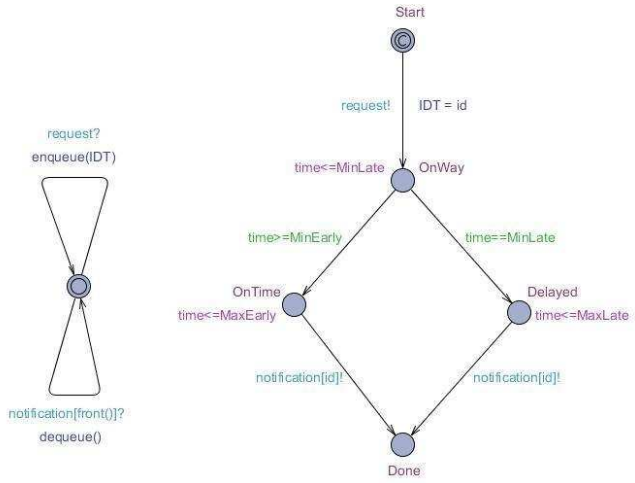

Figure 7. Crane-Truck automata.

To ensure the system correctness, we have formalized the required queries to verify certain properties by using the UPPAAL tool. The first property that we want to check is the absence of deadlocks in the model ($A[]$ *not deadlock*) and the second is the existence of an execution trace that allows all the truck automata to reach the state *done*, that is, all trucks accomplish their work ($E<>$ *Truck1.Done and Truck2.Done and Truck3.Done and Truck4.Done and Truck5.Done and Truck6.Done*). In Figure 8, the model checker obtains that the second formula is **satisfied**, so we ask UPPAAL to show us the trace that satisfies the formula, obtaining the trace depicted in Figure 9. Due to space limit, we only show the important part of this trace where we can see the necessary order of notifications to avoid the deadlocks in our model. Based on this we can ensure that the control tower needs to serve the trucks in this sequence: truck4, truck3, truck2, truck1, truck5 and truck6. Besides, the first formula is **not satisfied**, so we can ensure that the model has deadlocks. As we have found a design error in our model, we would have to go back to the design phase, correct the problem and repeat the process to check these properties again. The solution of this error is very simple since we have not taken into account the order of crane requests. The best way to solve the problem is by adding an incoming buffer to store the truck requests and sorting it according to the arrival time.

 ISBN: 978-1-61208-124-3

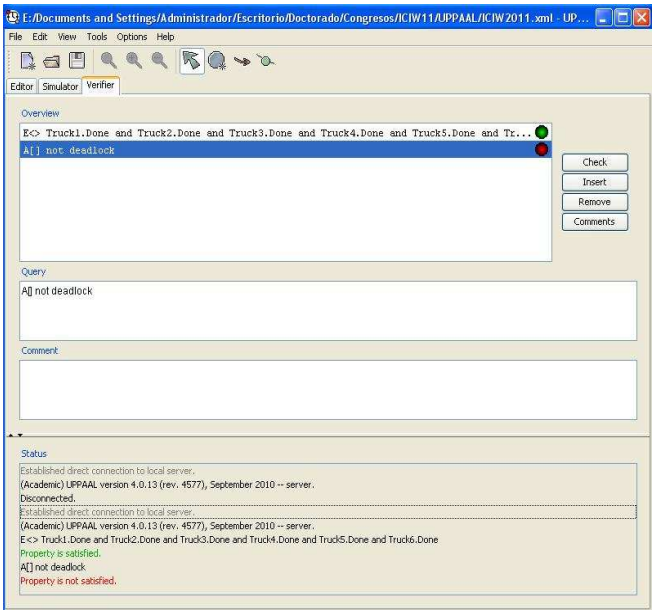

Figure 8. Screenshot of the UPPAAL verifier for Container terminal planning.

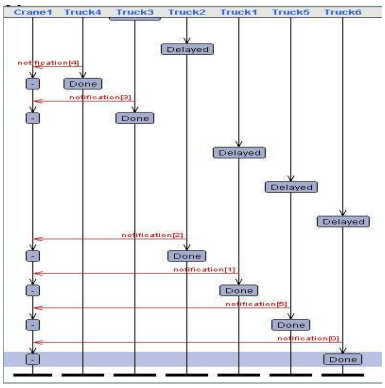

Figure 9. Screenshot of the UPPAAL simulator for Container terminal planning.

V. Conclusions and Future Works

Using formal methods is always beneficial to model, check and verify computer systems endowing these systems with mathematical rigour, minimum error rate and conformance with the specification. Moreover, adding stateful resources to Web services allows these services to store information that can be used in the future. This paper is a first approximation to the formal verification of WSRF specifications. Thus, we have shown how formal techniques, and in this specific case, timed automata, can be used to model and verify the use of resources in a Web services system. As future work, we are considering the possibility of implementing this model in Globus toolkit 4 by adding directly the queue system needed to ensure deadlock-freeness. Furthermore, we are working on a translation of WSRF into timed automata and Petri nets.

VI. Acknowledgement

Partially supported by the Spanish government (co financed by FEDER founds) with the project TIN2009-14312-C02-02 and the JCCLM regional project PEII09-0232-7745.

References

[1] T. Banks. *Web Services Resource Framework (WSRF) - Primer*. OASIS, 2006.

[2] M. E. Cambronero, G. Díaz, V. Valero, and E. Martínez. *Validation and verification of Web services choreographies by using timed automata*. Journal of Logic and Algebraic Programming, vol. 80(1), pp. 25-49, 2011.

[3] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MITPress, Cambridge, 1999.

[4] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. *THE WS-RESOURCE FRAMEWORK VERSION 1.0*. http://www.globus.org/wsrf/specs/ws-wsrf.pdf, 2004.

[5] G. Díaz, J. J. Pardo, M. E. Cambronero, V. Valero, and F. Cuartero. *Verification of Web Services with Timed Automata*. Notes in Theoretical Computer Science, vol. 157(2), pp. 19-34, 2006.

[6] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, T. Storey, and S. Weerawaranna. *Modeling Stateful Resources with Web Services*. Globus Alliance, 2004.

[7] A. Gudelj, M. Krcum, and Dragan Cisic. *Container Terminal Planning by Petri-net and Genetic Algorithms*. Proceedings of 10th International Conference on Traffic Science (ICTS), Transportation and globalization, 2006.

[8] D. Laforenza, R. Lombardo, M. Scarpellini, M. Serrano, F. Silvestri, and P. Faccioli. *Biological Experiments on the Grid: A Novel Workflow Management Platform*. 20th IEEE International Symposium on Computer-Based Medical Systems (CBMS'07), pp. 489-494, 2007.

[9] K.G. Larsen, P. Pettersson, and W. Yi. *UPPAAL in a Nutshell*. International Journal on Software Tools for Technology Transfer (STTT), 1997.

[10] T. Zang, R. Calinescu, S. Harris, A. Tsui, M. Kwiatkowska, J. Gibbons, J. Davies, P. Maccallum, and C. Caldas. *WSRF-Based Modeling of Clinical Trial Information for Collaborative Cancer Research*. IEEE International Symposium on Cluster Computing and the Grid, vol. 0, pp. 73-81, 2008.

[11] H. Ping and W. Xin-Lei. *Health Information System Grid Based on WSRF*. Second International Conference on Information Technology and Computer Science, pp. 518-521, 2010.

[12] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, and P. Vanderbilt. *Open Grid Services Infrastructure (OGSI) Version 1.0*, 2003.

 ISBN: 978-1-61208-124-3