Newcastle University e-prints

Date deposited: 10th September 2010


Peer Review Status: Peer reviewed

Citation for published item:

Further information on publisher website

Publishers copyright statement:

Use Policy:
The full-text may be used and/or reproduced and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not for profit purposes provided that:

- A full bibliographic reference is made to the original source
- A link is made to the metadata record in Newcastle E-prints
- The full text is not changed in any way.

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Robinson Library, University of Newcastle upon Tyne, Newcastle upon Tyne.

NE1 7RU.

Tel. 0191 222 6000
COMPUTING SCIENCE

Predicting Compliance of WSLA Contracts Using Automated Model Creation

Rouaa Yassin Kassab and Aad van Moorsel

TECHNICAL REPORT SERIES

No. CS-TR-1204       June 2010
Predicting Compliance of WSLA Contracts Using Automated Model Creation

R. Yassin Kassab, A. van Moorsel

Abstract

Service level agreements (SLAs) are designed to be monitorable contracts between a provider and consumer of a service. It is of interest to determine if such SLAs can also be used as input to a predictive model, e.g., a discrete-event stochastic model. For that purpose we study in this paper WSLA, the web service level agreement language, an SLA specification language tailored to web services. We provide a mapping from WSLA to metrics defined in the discrete event stochastic modeling formalism SDES (stochastic discrete event systems). We provide extensions to the commonly used SPNP tool to automate much of this process, such that given a WSLA document, the mapped SDES metrics can be derived automatically as much as possible. We will specify precisely what the modeler inputs during this process, and demonstrate the use of our tool for WSLA compliance prediction.
Abstract

Service level agreements (SLAs) are designed to be monitorable contracts between a provider and consumer of a service. It is of interest to determine if such SLAs can also be used as input to a predictive model, e.g., a discrete-event stochastic model. For that purpose we study in this paper WSLA, the web service level agreement language, an SLA specification language tailored to web services. We provide a mapping from WSLA to metrics defined in the discrete event stochastic modeling formalism SDS (stochastic discrete event systems). We provide extensions to the commonly used SPNP tool to automate much of this process, such that given a WSLA document, the mapped SDS metrics can be derived automatically as much as possible. We will specify precisely what the modeler inputs during this process, and demonstrate the use of our tool for WSLA compliance prediction.

About the author

Rouaa Yassin Kassab is working towards her PhD under supervision of Aad van Moorsel. Her research is in the area of business-driven system management based on quantitative models.

Aad van Moorsel joined the University of Newcastle in 2004. He worked in industry from 1996 until 2003, first as a researcher at Bell Labs/Lucent Technologies in Murray Hill and then as a research manager at Hewlett-Packard Labs in Palo Alto, both in the United States. Aad got his PhD in computer science from Universiteit Twente in The Netherlands (1993) and has a Masters in mathematics from Universiteit Leiden, also in The Netherlands. After finishing his PhD he was a postdoc at the University of Illinois at Urbana-Champaign, Illinois, USA, for two years. Aad has worked in a variety of areas, from performance modelling to systems management, from web services to cloud computing and on issues of security and trust. In his last position in industry, he was responsible for HP's research in web and grid services, and worked on the software strategy of the company. His research agenda aims at establishing an intelligent enterprise, with a specific focus on trust, privacy and security. The goal is to provide tools to improve IT decision making, if possible based on objective, quantitative methods, eventually fully automated. This involves mathematical modelling, algorithms and service-oriented software implementations. The recent research is highly interdisciplinary, using ideas from social and business sciences, to gain a deeper understanding of issues of trust in, for instance, cloud computing. DTI, EPSRC and EU-funded collaborations are ongoing with Hewlett-Packard, Merrill-Lynch, Warwick, Bath, UCL, various universities throughout Europe, and the Business School as well as the Medical School in Newcastle.

Suggested keywords

WEB SERVICES
SLA
TOOLS
SLA COMPLIANCE PREDICTION
Predicting Compliance of WSLA Contracts Using Automated Model Creation

Rouaa Yassin Kassab and Aad van Moorsel
School of Computing Science
Centre for Cybercrime and Computer Security
Newcastle University, UK
{rouaa.yassin-kassab, aad.vanmoorsel}@ncl.ac.uk

Abstract

Service level agreements (SLAs) are designed to be monitorable contracts between a provider and consumer of a service. It is of interest to determine if such SLAs can also be used as input to a predictive model, e.g., a discrete-event stochastic model. For that purpose we study in this paper WSLA, the web service level agreement language, an SLA specification language tailored to web services. We provide a mapping from WSLA to metrics defined in the discrete event stochastic modeling formalism SDES (stochastic discrete event systems). We provide extensions to the commonly used SPNP tool to automate much of this process, such that given a WSLA document, the mapped SDES metrics can be derived automatically as much as possible. We will specify precisely what the modeler inputs during this process, and demonstrate the use of our tool for WSLA compliance prediction.

1. Introduction

The increasing popularity of Internet and other computer services raises the need to clarify the relationship between the service provider and their customers regarding the overall quality of the offered service. For this reason, and to formalize the correlation of the system’s quantitative attributes with their desired values, a contract called Service Level Agreement (SLA) is established. This contract is used to identify the agreed quality measures, called the Service Level Objectives (SLO) [1], [2]. Furthermore, it usually specifies the penalties in case of contract violation.

SLA contracts could be written using different languages such as WSLA [3], WS-Agreement [4], and SLAng [5]. Each of these languages has its own syntax and semantics but they have in common declarations of several important pieces of information such as, the contractual parties, the quantitative attributes with their thresholds, and the penalties in case of contract breaching [6].

An SLA is typically defined in such a way that it is monitorable. For instance, in the work presented in this paper we will concentrate on WSLA, the web service level agreement language [3], which specifies in detail at which moments during the service run time the metrics should be observed. This provides the opportunity to automate the deployment and configuration of monitoring software based on the SLA specification, as pursued in [1]. The authors in [2] take this a step further and assume the SLA to be a software object that can be dynamically accessed, read and be written during its life time.

In this paper we pursue yet another use of the SLA, namely that as the specification of metrics for a predictive discrete-event stochastic model. Service providers want to be able to predict the expected level of compliance before agreeing to an SLA with a customer, and the automated conversion from an SLA to a metric in a discrete event stochastic model is therefore of interest. To the best of our knowledge this use of an SLA has not yet been considered in literature. However, the use of an SLA for predictive modelling is not necessarily trivial since SLAs are written for the purpose of monitorability, not for model-based prediction. As an example, a steady-state metric one will not find in an SLA, and instead a function over periodically monitored variables is defined. This makes the current work fundamentally different from the proposal in [7] to use model-based metric definitions as SLAs.

Also related to this paper is the work such as in [8] in which model-based prediction of SLA compliance is carried out. In that and similar work, however, the SLA language is chosen to be a language that defines metrics that fit with the stochastic processes considered. In this work, we research the suitability for an existing SLA language to be ‘transformed’ into metrics of a discrete-event stochastic model, as much as possible in automated fashion. We note that in our previous work [9] we proposed the same idea, but only presented an ad hoc mapping from WSLA to
Möbius rewards [10]. The work in this paper is much more generic, providing a mapping to a stochastic process definition instead of tool syntax, and providing tool support that automates the mapping as much as possible.

In this paper we map WSLA on metrics in SDES, the stochastic discrete event system formalism developed in [11]. We choose to use the IBM’s web service level agreement language [3] for no particular reason other than that it is published and publicly available. Similarly, we could have used alternatives for SDES. When carrying out the mapping from WSLA to SDES metrics, we assume certain semantics for WSLA elements to establish the mapping unambiguously. It will turn out that we cannot automate all aspects of the mapping process, since the WSLA document does not provide information about the system dynamics itself. This implies that a model must be built independently by the modeler, and also means that during the translation process from WSLA to SDES, the modeler is required to input state variables or actions pertaining to the SDES metric.

We develop tool support for WSLA compliance prediction, automating the mapping process from WSLA to SDES metrics, by augmenting SPNP [12]. We use stochastic reward nets in SPNP, which are a special case of SDES. The WSLA compliance prediction tool parses the WSLA document several times to obtain the SPNP code corresponding to the WSLA document under consideration. It also provides a user interface to assist the modeler in adding the needed info to complete the translation process.

The rest part of the paper is organized as follows. In Section 2 we present the relevant background information. In Section 3 we outline our approach, while in Section 4 we will present the detailed mapping from WSLA to SDES metrics. An implementation and a detailed running example will be introduced in Section 5. Finally, we conclude in Section 6.

2. Background

In this paper we map WSLA service level agreements to reward metrics in a discrete-event dynamic system. For the latter, we therefore also require a formalism, to describe the system dynamics as well as the metrics defined in the service level agreement. In general terms, we use the approach of Sanders to define the space of possible metrics [10], and use Zimmerman’s Stochastic Discrete Event System (SDES) formalism to describe the stochastic process. In this section, we introduce the target formalism SDES and the origin formalism WSLA, respectively.

2.1. Stochastic Discrete Event System

We map the service level agreements on reward metrics in a discrete-event dynamics system. We could have chosen from a multitude of general formalisms to specify stochastic process, but chose Zimmerman’s Stochastic Discrete Event Systems (SDES) [11]. Formalisms such as stochastic Petri nets or queuing networks are special cases of SDES, and SDES is thus a powerful enough formalism to map on any of these special cases.

An SDES is expressed as a stochastic process $SP(X(t), t \in T)$, where $T$ is time. SDES states are described by assigning a value to each State Variable and the process moves between states through an Action in the set of actions $A$. So, in the context of a stochastic Petri net, the states are determined by the number of tokens in all states, and actions are firing transitions. The SDES defines a Sort $S$ which returns all the values the state variables can take. So, formally we have the following definition for SDES:

**Definition 1:** A stochastic discrete event system (SDES) is defined as the following tuple [11]:

$$(SVA,S,RV),$$

where:

- $S$ : the set of all state variables
- $A$ : the set of actions
- $S$ : all possible values of a state variable $sv \in S V$
- $RV$ : set of all reward variables

Each reward variable $rv \in RV$ maps the stochastic process $SP$ to a real value, that is:

$$rv : SP \rightarrow \mathbb{R}$$

for all $rv \in RV$.

For our purposes, we need to further define reward variables, that is, the elements in $RV$:

**Definition 2:** Each reward variable $rv \in RV$ is a tuple:

$$rv = (rv_{rate}, rv_{imp}, rv_{int}, rv_{avg}),$$

where:

- $rv_{rate}$ is the rate reward, gained in a state $sv \in SV$ of the model over time
- $rv_{imp}$ is the impulse reward, gained when a specific action fires
- $rv_{int} : [lo, hi]$ with $lo, hi \in \mathbb{R}_{0^+} \cup \{\infty\}$ is the observation interval under consideration, specified by the boundaries $lo, hi$. For example, instant of time equates to $lo = hi$ and interval of time implies $lo < hi$.
- $rv_{avg}$ is a boolean specifying if the resulting measures should be computed as an average over time ($rv_{avg} = TRUE$) or accumulated ($rv_{avg} = FALSE$).

As examples, an instant of time distribution is captured by setting the parameters $rv_{int} = [t, t]$ and $avg = FALSE$. The average accumulated reward up until time $t$ is set through $rv_{int} = [0, t]$ and $rv_{avg} = TRUE$, etc.
2.2. WSLA

Web Service Level Agreement (WSLA) is an SLA specification language in XML tailored to web services [3]. Here we review only the main elements of WSLA important to our work. For full details, we refer to [3].

Table 1 provides an illustrative example of a WSLA document. A WSLA document consists of three main sections: Parties, ServiceDefinition and Obligations. The Parties section contains all the information about the parties that are involved in the contract. The ServiceDefinition section contains a description of all the metrics. The Obligations section contains the actual Service Level Objectives, defined through thresholds for the described service definition attributes, as well as the consequences of contract violations [3]. We describe in more detail the service definition and obligations, since these contain the quantitative attributes and their thresholds.

2.2.1. Service Definition. ServiceDefinition contains two main elements: Schedule and Operation. We explain Operation first, since it defines the metrics—Schedule then defines the time interval over which these metrics need to be collected.

The term Operation points to the fact that WSLA is meant to be used for web services. We do not exploit this fact in the current derivation of the model and metrics, since we are primarily interested in automating the definition of service level objectives. These service level objectives are represented as SLAParameter in WSLA, which in turn contain a representative Metric element.

At its core, each Metric that defines a SLA parameter has a MeasurementDirective, which refers to the status, the throughput, or generic counters or gauges (for example, Gauge in Metric ProbedUtilization). Section 4 provides a mapping of all available measurement directives on a reward construct. Such a measurement directive can then be built up into a full blown service level objective by composition and by using functions. For instance, Metric UtilizationTimeSeries defines a time series of 12 elements of the ProbedUtilization metric. In turn, Metric OverloadPercentageMetric applies a function on that time series.

Note that OverloadPercentageMetric and UtilizationTimeSeries make use of the Schedule elements which are specified in the same ServiceDefinition section. The schedule specifies the interval of time or the instants of time at which the value of the metrics should be collected and computed. Schedule contains Period and Interval subelements, which specify the time through which the function is applicable (day, month, etc.) and the instants of time a new value is obtained (every minute, hour, etc.).

Table 1. WSLA example.
We see from the example WSLA document that WSLA is designed to define the monitoring requirements for measuring service level objectives. That is, it is not aimed at modelling. As an example, time series of 12 elements, and 5-minute schedules differ from the typical subdivision of metrics found in the modelling literature: interval-of-time versus instant-of-time, steady-state versus transient, etc. Similarly, the thresholds defined in the obligations part correspond to distribution metrics, thus complicating (or making superfluous) the averaged metrics often relevant in modelling problems. We will see in what follows what the consequences are of this inherent tension between a monitoring focused SLA specification language such as WSLA and off-line model-based analysis.

2.2.2. Obligations. The Obligation part defines the actual service level objective, by setting the threshold for each SLA parameter and the applied penalties in case of contract breaching. Each Obligation contains an Expression element with a SLAParameter, associated border Predicate and Value. A service level objective could have more than one expression, each using a different SLAParameter as target. These expressions can be joined using logical operators such as AND, NOR, etc.

3. Outline of the Approach

The approach we take in this paper is given in Figure 1. It consists of seven steps, which we will explain in what follows. The core of the approach is the creation of an Engine, the dark gray box in Figure 1, which automates as much as possible the process of WSLA compliance prediction. Note that solid boxes and arrows denote fully automated aspects, dotted boxes denote semi-automated aspects and dashed boxes and arrows denote non-automated aspects. All automated and semi-automated parts are implemented in our WSLA compliance prediction tool, as we will explain in Section 5.

Step 1: Create the Base Model. As we remarked, WSLA documents do not provide information about the dynamics of the system, and, as a consequence, a modeler must model the dynamics of the system independently, based on knowledge of the system. In essence, a modeler can use any appropriate SDES modelling approach. This step cannot be automated. Moreover, throughout the compliance prediction process, the modeler will be asked to provide the necessary information to the engine so that rewards will be assigned correctly, depending on the specific model. This will be explained in Step 4.

Step 2: Import a WSLA Document. A WSLA document is created for the system or the next customer. This is used as an input for our Engine (dashed line WSLA document in Figure 1).

Step 3: General Reward Specification Engine. The GRS engine translates the measurement directive in the WSLA document in SDES reward functions. This step is fully automated, and results in a general specification of rewards including information about the time domain at which the measure should be computed, as defined through the metrics.
Step 4: Translator Engine. The TS engine translates the general reward specification into one that is compatible with the SDES model used. For instance, in our tool we use reward nets, based on the SPNP software. As we noted, this step is semi-automated, since the modeler is needed to provide the connection between rewards and SDES constructs. Through a dedicated GUI (‘User Interface’ in the figure), the modeler will choose the suitable SDES primitives that correspond to the state variables or actions in the general reward specification.

Step 5: Solve the Model. Solve the model using solvers and tools associated with the SDES of choice. As we mentioned, in our work we use SPNP’s algorithms as solvers. Note that the resulting results are not yet equivalent to the WSLA specification, as we will see in the next step.

Step 6: Result Accumulation and Comparison Engine. The RAC engine applies the remaining WSLA-specified computations to the solver results. WSLA specifies functions inside composite metrics to compute the actual value of a SLAParameter. The functions aggregate, average, sum or otherwise manipulate solver results. This step is fully automated, and the ‘Final Result’ in the figure is shown through a GUI associated with the RAC engine.

Step 7: Determine Compliance. Finally, in Step 7, compliance of the WSLA contract is determined based on the final results. The RAC GUI can automatically display comparison results in an intuitive manner.

4. Mapping from WSLA to SDES Metrics

In this section we will describe the operation of the three components of the Engine in Section 3. As we remarked in Section 2.2, at the core of any service level objective in WSLA is the measurement directive. In this section we provide an unambiguous mapping from any measurement directive to SDES. We point out where the modeler needs to input information to the Translator Engine to complete the mapping between WSLA and SDES.

Table 2 provides the generic structure of a metric and the associated measurement directive. WSLA defines seven different measurement directive types, namely InvocationCount, Status, StatusRequest, ResponseTime, DownTime, Counter and Gauge [3]. The type is specified in the type attribute of the MeasurementDirective element. The type of the measurement affects the result type specified in the resultType attribute, which in turn affects the type of the metric that encloses it. So, the values in italics in Table 2 change depending on the measurement type, all other tags remain the same. In any case, we provide a translation for all seven measurement directives in what follows.

Table 2. Measurement directives in WSLA

4.1. InvocationCount

WSLA defines InvocationCount as “the number of usages of an operation” [3]. In other words, it corresponds to the throughput of a service. Its WSLA syntax is as follows:

```xml
<Metric
    name="Metric_name"
    type="Metric_type"
    unit="Metric_unit">
    <MeasurementDirective
        xsi:type="wsla:MeasurementDirective_type"
        resultType="MeasurementDirective_result_type"
        ...</MeasurementDirective>
</Metric>
```

The natural manner to derive this metric from an SDES is to associate an impulse reward to an action \(a_i \in A\):

\[
rv_{var} = \begin{cases} 
rv_{rate}(sv) = 0 & \forall sv \in SV \\
rv_{imp}(a) = \begin{cases} 
1 & \text{if } a = a_i \\
0 & \text{otherwise}
\end{cases} 
\end{cases}
\]

The remaining issue is, which action to chose. For this purpose, the Translator Engine asks the modeler to provide the name of the action. When deploying this mapping in our tool, the modeler should choose which actions are related to the particular operation in order to produce the desired throughput. We write this down as:

Input modeler: action \(a_i\).

4.2. StatusRequest and Status

StatusRequest and Status will give 1 if the system is up and 0 otherwise [3], with as only difference that StatusRequest returns an integer and Status returns a boolean, a difference that is not important for compliance prediction. Both follow the syntax in Table 2 with the specific measurement directive definition for StatusRequest as follows:

```xml
<MeasurementDirective
    xsi:type="wsla:StatusRequest"
    resultType="integer"/>
```

Assume that \(\sigma(sv) \in S(sv)\) is the value of the state variable \(sv\), and \(specific\_values \subseteq S(sv)\) is a set of specific values under which this state variable \(sv\) is considered to bring the system down, then the corresponding reward function is:

\[
rvar = \begin{cases} 
rv_{imp}(a) = 0 & \forall a \in A \\
rv_{rate}(sv) = \begin{cases} 
1 & \text{if } \sigma(sv) \in specific\_values \\
0 & \text{otherwise}
\end{cases}
\end{cases}
\]
The modeler needs to chose the appropriate state variables and associated values corresponding to system down status.

**Input modeler**: state variable \( sv \) and set specific values that gives status 0.

In the tool (see Section 5) we provide a particular solution to specify the set specific values, which is less generic but easier to use. The modeler chooses a state variable, one value for 'specific value', and a relational symbol \((\prec,\succ,=,\preceq,\succeq)\). The tool then derives the set specific values and will also aid the modeler by listing all the state variables available for the modeler to chose from.

### 4.3. ResponseTime

ResponseTime denotes the time between sending a request and receiving its response. The syntax is as follows:

```xml
<MeasurementDirective	xsi:type="wsla:ResponseTime"
resultType="double">
</MeasurementDirective>
```

To express the response time in term of rewards, we use an additional state variable \( sv \in SV \) which signals the receipt of the response. The state variable \( sv \) is initially set to 0 and can jump to 1 once only, indicating the response has been received. Then, \( R(t) \), the probability that the response time is less than \( t \) is equal to the probability that the state variable \( sv \) is 1 at time \( t \). So, we determine the response time \( R \) of an operation by checking at each time instance if the state variable equals 1. That is:

\[
R(t) = Pr(response\ time \leq t) = Pr(sv = 1 \ at\ time\ t)
\]

This is represented using a rate based reward function as follows:

\[
rv_{var} = \begin{cases} 
rv_{imp}(a) = 0 & \forall a \in A \\
rv_{sv}(sv) = 1 & \text{if } \sigma(sv) = 1 \\
0 & \text{otherwise}
\end{cases}
\]

The response time distribution is then computed by determining the expected reward at time \( t \). This leaves us with one complicating factor, well-known when computing response times: the response time computed in above manner depends on the initial state. Often it is appropriate to take the steady-state distribution as initial state, but this depends on the circumstances. Our tool leaves it to the modeler to set an appropriate initial state. This is not completely satisfactory, since it requires good modeling judgment by the modeler, but we have not yet developed a way to avoid this. The modeler’s job thus includes the following:

**Input modeler**: Introduce a state variable \( sv \in SV \) in the model that is set to value 1 when the response is given, and determine an appropriate initial state for the model.

### 4.4. DownTime

The DownTime measurement directive gives a direct reading of the total time through which the system is considered at a down status [3], with as syntax:

```xml
<MeasurementDirective
xsi:type="wsla:DownTime"
resultType="double">
</MeasurementDirective>
```

As a consequence, the mapping is identical to Status, but measured as an interval-of-time metric, not an instant-of-time metric.

### 4.5. Counter

Counter according to WSLA “describes the relevant information to retrieve a counter from the instrumentation of a service or managed resource” [3] and it is used to count specific events of a service. Counter generalizes InvocationCount to counters that are not related to the operation occurrences but also other actions. We map Counter on SDES in the same way as InvocationCount.

```xml
<MeasurementDirective
xsi:type="wsla:Counter"
resultType="integer">
</MeasurementDirective>
```

\[
rv_{var} = \begin{cases} 
rv_{var}(sv) = 0 & \forall sv \in SV \\
riv_{imp}(a) = 1 & \text{if } a = a_i \\
0 & \text{otherwise}
\end{cases}
\]

**Input modeler**: action \( a_i \).

### 4.6. Gauge

Gauge is defined in WSLA as “a non-negative integer that may increase or decrease, and is used to measure the current value of some entity” [3]. In essence, this corresponds to the value of a state variable in the system, and in SDES terms we can allow rate as well as impulse rewards to add to the Gauge.

Let us first give the WSLA snippet representing a gauge:

```xml
<MeasurementDirective
xsi:type="wsla:Gauge"
resultType="double">
</MeasurementDirective>
```

In the translation to SDES, we can provide a definition that allows any rewards to be added to the gauge. The reward definition is then unrestricted, and the modeler needs to assign which rewards are added to the gauge. We also provide a special gauge, corresponding to a single state variable representing the gauge value. Depending on the model at hand, this simplifies the job of the modeler. Assume a state variable \( sv_i \in SV \),
with \( \sigma(sv_i) \) is the value this state variable, then the reward variable is defined as:

\[
rv_{var} = \begin{cases} 
\frac{rv_{int}(a)}{a \in A} & \text{if } sv = sv_i \\
\sigma(sv) & \text{otherwise}
\end{cases}
\]

Input modeler: choose or introduce state variable \( sv_i \).

4.7. Mapping Monitoring Times

Of particular interest is the mapping of the monitoring times (specified in the Period tags of WSLA) on the model-based reward metrics defined by Sanders [10]. WSLA specifies the times at which one monitors for a metric, to determine whether the service level objectives have been met or not. One can use exactly these instances of time for the model-based prediction as well, as we will do here, and it will translate into a series of transient results one needs to obtain from the model. However, it should be noted that such time dependent metrics make the model typically less tractable than if one would use a steady-state measure, and we therefore think that for predictive model-based SLA assessment steady-state measures are often a reasonable choice. In the current paper we will not consider this option, but instead literally translate the time instances defined in WSLA into SDES.

The specification of time instances in WSLA is as follows, with the top part of Table 3 showing the Schedule specification (with its subelements Interval and Period) and the bottom part showing its use in a Metric definition:

```xml
<Schedule name="5minuteschedule">
  <Period>
    <Start>2001-11-30T14:00:00.000-05:00</Start>
    <End>2001-12-31T14:00:00.000-05:00</End>
  </Period>
  <Interval>
    <Minutes>5</Minutes>
  </Interval>
</Schedule>
```

\[ \text{Metric name="UtilizationTimeSeries" type="TS" unit=""/>\]
\[ <Function xsi:type="TSConstructor" resultType="float">
  <Schedule>5minuteschedule</Schedule>
  <Metric>ProbedUtilization</Metric>
  <Window>12</Window>
</Function>
```

Table 3. WSLA structure for monitoring times

The role of Schedule is to specify the times at which the metrics are collected, which is done according to different instants of time (Interval element) over a particular interval of time (Period element). To map this on SDES, we define \( s \) a series of instances of times as follows:

For \( (i = 0; i \leq \text{Period}; i = i + \text{Interval}) \)

compute \( rv_{var} \), where \( rv_{int} = [i, i] \) and \( \text{avg} = \text{FALSE} \).

5. Implementation and Case Study

In this section we will apply the general solution presented in Section 3 to a system modeled using one specific SDES model, namely Stochastic Reward Networks (SRN) developed by [13]. The modeling tool that is used to build and solve this model is the Stochastic Petri Net Package (SPNP) tool [12] and our implementation of the engine is built in Java using the Eclipse platform. We first describe the system in Section 5.1 and present an associated WSLA contract as well as the SRN model of this system. The working of our tool is described in Section 5.2.

5.1. System Description

The system we are interested is taken from Ma, Ro and Trivedi [14], and is chosen rather arbitrarily. It is a cellular network with channel recovery scheme for fixed channel assignment. In this system, the typical model assumption is made that all cells are alike, so one cell can illustrate the behavior of the whole system [14]. This system (presented as an SRN in Figure 2) consists of one cell served by a set of channels pooled in a channel pool (with a total of \( C \) channels). While the channels are in the pool, they are idle and ready to serve the incoming calls.

There are two kinds of calls which are the New Call (NC) and the Handoff Call (HC) and they are treated differently. The Base Station (BS) in the cell will preserve \( C_h \) of its idle channels for HC: the call of this type is connected as long as an idle channel exists. However, in the case of an NC, the call will be blocked if the channel pool has less than \( C_h \) idle channels (that is, more than \( C - C_h \) channels are engaged) [14]. A Failed Call (FC) happens if the channel that services this call breaks down. The call will be reinstated immediately if an idle channel exists, irrespective of whether the call is NC or HC. If the channel pool is empty, the failed call will be queued until an idle channel exists. A non-failed call ends either because of call completion, which is modelled through an exponentially distributed duration with rate \( \lambda_d \) or because of the Mobile Station (MS) leaving the current cell for another one (occurs with rate \( \lambda_b \)). In these cases the channel will return to the channel pool [14]. As is standard in this type of models, a fixed point iteration schema is used to obtain the HC arrival rate for the cell.

WSLA Contract. Let us assume, the provider of this network offers a services with high availability. In particular, we assume the SLA consists of one service level objective, which says that channel availability is higher than 96\%. In Table 4, we present a simple WSLA contract for our system.
<xml version="1.0"/>
<SLA xmlns="http://www.ibm.com/wsla"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/wsla WSIA.xsd"
name="Cellular_Networks">
  <Parties>
    <ServiceProvider name="ServiceProvider">
      <Contact>
        <Person>network_provider</Person>
      </Contact>
    </ServiceProvider>
    <ServiceConsumer name="ServiceConsumer">
      <Contact>
        <Person>network_consumer</Person>
      </Contact>
    </ServiceConsumer>
  </Parties>
  <ServiceDefinition name="Cellular_Service">
    <Schedule name="availability_schedule">
      <Period>
        <Start>2009-06-01T13:00:00.000-05:00</Start>
        <End>2009-06-01T14:00:00.000-05:00</End>
      </Period>
      <Interval>
        <Minutes>1</Minutes>
      </Interval>
    </Schedule>
    <Operation xsi:type="WSDLSOAPOperationDescriptionType" name="Operation">
      <SLAParameter name="channel_availability" type="float" unit="percentage">
        <Metric>availability</Metric>
      </SLAParameter>
      <Metric name="availability" type="float">
        <Function xsi:type="wsla:Divide">
          <Operand>
            <Metric>up OCCurrence</Metric>
          </Operand>
          <Operand>
            <LongScalar>60</LongScalar>
          </Operand>
        </Function>
      </Metric>
      <Metric name="up OCCurrence" type="long">
        <Function xsi:type="wsla:ValueOccurs" resultType="long">
          <Metric>recorded_status</Metric>
          <Value>
            <LongScalar>1</LongScalar>
          </Value>
        </Function>
      </Metric>
      <Metric name="recorded_status" type="TS">
        <Function xsi:type="wsla:TSConstructor" resultType="TS">
          <Schedule>availability_schedule</Schedule>
          <Metric>network_status</Metric>
          <Window>60</Window>
        </Function>
      </Metric>
      <Metric name="network_status" type="integer">
        <MeasurementDirective xsi:type="wsla:StatusRequest" resultType="integer"/>
      </Metric>
    </Operation>
  </ServiceDefinition>
  <Obligations>
    <ServiceLevelObjective name="availability SLO">
      <Expression>
        <Predicate xsi:type="Greater">
          <SLAParameter name="channel_availability">
            <Value>96</Value>
          </SLAParameter>
        </Predicate>
      </Expression>
    </ServiceLevelObjective>
  </Obligations>
</SLA>

Table 4. The WSLA contract for the system.

SRN model of the system using SPNP. We assume the modeler uses Stochastic Reward Net (SRN) to construct a model, and for this discussion we simply use the model from [14]. This model is built using the Stochastic Petri Net Package (SPNP) tool [12]. Figure 2 provides the graphical version of the model. We will not explain the details of an SRN and refer to [13].

For our tool, we also require the textual SPNP interface for model definition, since we eventually map the SDES and WSLA functions on SPNP textual code. For this, SPNP uses a CSPL file (C-based language specific to SPNP), which, for ease of reference, we will call the ‘SPNP file’. Table 5 provides the CSPL version of the same model as Figure 2. The SPNP file contains a description of the SRN model of the system with the SPNP functions that are necessary to solve the model. Using the SPNP GUI we produce an SRN model graphically, which will automatically produce the SPNP file with all the CSPL functions available. The engines we implemented then inserts automatically the correct code snippets into this SPNP file.

5.2. WSLA Compliance Prediction Tool

The SPNP file used as input for our WSLA compliance prediction tool contains only the description of the SRN model, and leaves empty the SPNP functions assert, ac_init, ac_reach and ac_final. Our tool then automatically inserts the reward functions and the other translated WSLA functions in the appropriate positions in the SPNP file. The tool parses the WSLA document and makes a recursive call to identify MeasurementDirective belonging to each SLAParameter. Depending on the type of this measurement, the tool will automatically present an interactive GUI that allow the modeler to choose the primitives that satisfies this kind of measurement, presenting only the suitable set of primitives (places and transitions) available in the model for this measurement. For example if the measurement type requires a place selection (e.g., for ResponseTime), the tool present a GUI that lists all places available in the model to select from.

Based on the parsing of the WSLA file, and the input from the modeler, a reward function written in CSPL will be produced and embedded automatically in the SPNP file. The tool then also automatically generates
#include <stdio.h>
#include <math.h>
#include "user.h"

/* global variables */
int t_channel = 28; lam_h_i = 0.2; double lam_n = 10;
double mu_r = 6.8107; lam_h_o = 0.11; lam_d = 0.5;
double lam_f = 0.000016677; Int g = 1;

/* =================================================================================
* DEFINITION OF THE NET
* =================================================================================*/
void options()
{
    /* Insert Arcs */
    iarc("t1","T"); iarc("t1","R"); iarc("t1","B"); iarc("t1","CP");
    oarc("t1","T"); oarc("t1","CP"); moarc("t1","R");
    parm("lam_h_i"); parm("lam_n");
    place("T"); place("B"); place("R"); place("CP");
    init("CP",t_channel);
}

/* =================================================================================
* TRANSITION DEFINITION
* =================================================================================*/
void ac(
    void assert()
{
    /* Compute occurrences of a specific value. */
    if ( mark("CP") > @ ) returning
        return (1.0); else return (0.0); }

/* Information on the reachability graph */
for ( loop=0; loop
    solve ((double) loop);
    value_set[index] = expected(StatusRequest_RF());
    index"="index+1; }

Table 7. SPNP input for WSLA’s TSConstructor.
After creating the reward function that corresponds to the measurement directive, the tool continues in this recursive call to use the value derived from this reward function as an input to the function that calls it. In our example, the function that calls StatusRequest is the TSConstructor. In general, this function stores the values taken from solving the reward function in an array of size equal to the value of the window element and through a schedule (represented by the Schedule element) during a specific time span (the value of Period element) and at specific instances (the value of Interval element). This means that the time mapping for this reward function will be computed at a sequence of instants of time starting from 0, increased by the Interval value, until the upper bound Period value, as we explained in Section 4.7. The resulting CSPL for the previous function is shown in (Table 7). It contains all the necessary variable definitions and the corresponding code to all of the subelements of the TSConstructor function.

double ValueOccurs =0.0; int i_Occur@0;
/* Compute occurrences of a specific value. */
for ( i_Occur@0; i_Occur<@; i_Occur++)
    ValueOccurs = ValueOccurs + value_set[i_Occur]; }

Table 8. SPNP input for WSLA’s ValueOccurs.
The next element to consider is ValueOccurs, which depends on the value produced from the TSConstructor. This function counts the number of occurrences of a specific value inside the array. Here in our example, the function should count how many times the system is up (the value equals 1). In other words how many times the value 1 appears in the array value_set[]. Because we are dealing with probabilities, the array contains float values (not the integer values 0 or 1). So, we sum all the value stored in the array (60 values in our case according to the window of the TSConstructor). The CSPL code generated automatically can be found in Table 8.

The final WSLA function used for computing the result of the SLAParameter is the Divide function. It
When the tool calls the SPNP solver, it generates the value for the \texttt{Divide} function. The tool will also add a new SPNP function called \texttt{Divide} which is the SPNP function which is \texttt{pr_value()} to generate the expected value of the last result taken from the Divide function (Table 10).

### Solving the Model

When the tool calls the SPNP solver, it generates the value for the SLAPParameter, called \texttt{Channel\_availability}. Our WSLA compliance prediction tool will compare the results with the SLO value and informs the user if it meets the target of 96%. To illustrate the results we obtain with the tool, recall that incoming \texttt{NC} calls will be blocked if the channel pool has less than \texttt{C_h} idle channels. So, the value of \texttt{C_h} will affect the availability of channels. For example, if \texttt{C_h} = 1, the condition will be that the channel should at least have two idle channels to be able to serve an incoming new call and so on. Table 11 present different value of \texttt{C_h} and the availability condition for each of them and how this affects the expected availability percentage.

### 6. Conclusion

In this paper we investigated if SLA definitions can be used as specification of reward metrics in discrete-event stochastic models. We did so for WSLA, the web service level agreement language, and found that the process of translating from WSLA to a stochastic model can be partially automated. The challenge lies in the fact that SLA definition languages are designed to be used for monitoring, not model-based prediction, as well as in the fact that the SLA does not provide much information about the system operation itself. As a consequence, at times the modeler needs to manually provide the necessary information. We have built the WSLA compliance prediction tool as an extension to SPNP that facilitates the translation process from WSLA to reward nets, and provides interactive graphical user interface assistance for the modeler’s input.

### References


