SERSCIS-Ont

A Formal Metrics Model for Adaptive Service Oriented Frameworks.

Mike Surridge, Ajay Chakravarthy, Maxim Bashevoy, Martin Hall-May
IT Innovation Centre, 2 Venture Road, Chilworth, Southampton, UK.
{ms,ajc,mvb,mhm}@it-innovation.soton.ac.uk

Abstract— **In the Future Internet, programs will run on a dynamically changing collection of services, entailing the consumption of a more complex set of resources including financial resources. The von Neumann model offers no useful abstractions for such resources, even with refinements to address parallel and distributed computing devices. In this paper we detail the specification for a post-von Neumann model of metrics where program performance and resource consumption can be quantified and encoding of the behaviour of processes that use these resources is possible. Our approach takes a balanced view between service provider and service consumer requirements, supporting service management and protection as well as non-functional specifications for service discovery and composition.**

Keywords-adaptive metrics; SOA; measurements;constraints; QoS

I. INTRODUCTION

A (relatively) open software industry developed for non-distributed computers largely because of the von Neumann model [8], which provided the first practical uniform abstraction for devices that store and process information. Given such an abstraction, one can then devise models for describing computational processes via programming languages and for executing them on abstract resources while controlling trade-offs between performance and resource consumption. These key concepts, resource abstraction supporting rigorous yet portable process descriptions, are fundamental to the development and widespread adoption of software assets including compilers, operating systems and application programs.

In the Future Internet, programs will run on a dynamically changing collection of services, entailing the consumption of a more complex set of resources including financial resources (e.g. when services have to be paid for). The von Neumann model offers no useful abstractions for such resources, even with refinements to address parallel and distributed computing devices. In this context, we need something like a 'post-von Neumann' model of the Future Internet of Services (including Grids, Clouds and other SOA), in which: program performance and consumption of resource (of all types) can be quantified, measured and managed; and programmers can encode the behaviour of processes that use these resources, including trade-offs between performance and resource consumption, in a way that is flexible and portable to a wide range of relevant resources and services.

In this paper, we describe the metric model developed within the context of the SERSCIS project. SERSCIS aims to develop adaptive service-oriented technologies for creating, monitoring and managing secure, resilient and highly available information systems underpinning critical infrastructures. The ambition is to develop technologies for such information systems to enable them to survive faults, mismanagement and cyber-attack, and automatically adapt to dynamically changing requirements arising from the direct impact of natural events, accidents and malicious attacks. The proof of concept (P-o-C) chosen to demonstrate the SERSCIS technologies is an airport-based collaboration and decision-making scenario. In this scenario, separate decision makers must collaborate using a number of dynamic interdependent services to deal with events such as aircraft arrival and turn-around, which includes passenger boarding, baggage loading and refuelling. The problem that decision makers face is that the operations are highly optimised, such that little slack remains in the turnaround process. If a disruptive event occurs, such as the late arrival of a passenger, then this has serious knock-on effects for the rest of the system that are typically difficult to handle.

The focus for our work is therefore to support the needs of both service providers and consumers. Our goal is to allow providers to manage and protect their services from misbehaving consumers, as well as allowing consumers to specify non-functional requirements for run-time service discovery and composition should their normal provider become unreliable. In this sense, SERSCIS-Ont combines previous approaches from the Semantic Web community focusing on service composition, and from the service engineering community focusing on quantifying and managing service performance.

The rest of the paper is organised as follows. Section II defines and clarifies the terminology used for metrics, measurements and constraints. In Section III we present the SERSCIS-Ont metric model. Here each metric is discussed in a detail along with the constraints which can be imposed upon these metrics. Section IV reviews the state of the art for related work and compares and contrasts research work done in adaptive system metrics with SERSCIS-Ont. Section V presents the results of the validation/simulation experiment carried out to test the applicability of the SERSCIS metrics. Finally we conclude the paper in Section VI

II. METRICS MEASUREMENTS AND CONSTRAINTS

It is important to distinguish between the terminology used for metrics, measurements and constraints. In Figure **1** we show the conceptual relationships between these terms.

 ISBN: 978-1-61208-109-0

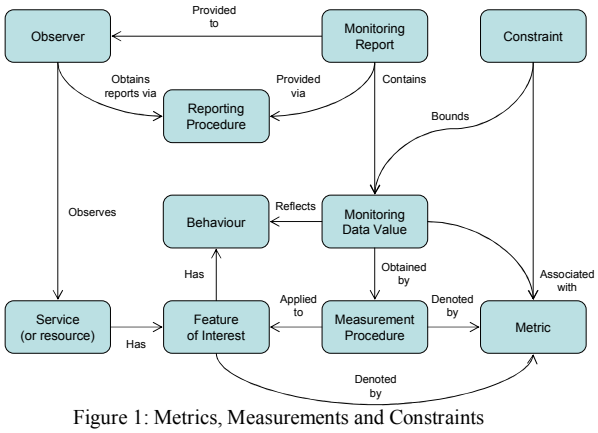

Figure 1: Metrics, Measurements and Constraints

Services (or sometimes the *resources* used to operate them) are monitored to provide information about some *feature of interest* associated with their operation. The *monitoring data* by some *measurement procedure* applied to the feature of interest at some time or during some time period. *Metrics* are labels associated with this data, denoting what feature of interest they refer to and (if appropriate) by which measurement procedure they were obtained. Finally, monitoring data is supplied to *observers* of the service at some time after it was measured via *monitoring reports*, which are generated and communicated to observers using a *reporting procedure*. It is important to distinguish between monitoring data for a feature of interest, and its actual *behaviour*. In many situations, monitoring data provides only an approximation to the actual behaviour, either because the measurement procedure has limited accuracy or precision, or was only applied for specific times or time periods and so does not capture real-time changes in the feature of interest. *Constraints* define bounds on the values that monitoring data should take, and also refer to metrics so it is clear to which data they pertain. Constraints are used in *management policies*, which define management actions to be taken by the service provider if the constraints are violated. They are also used in *SLA terms*, which define commitments between service providers and customers, and may specify actions to be taken if the constraints are violated. Note that management policies are not normally revealed outside the service provider, while SLA terms are communicated and agreed between the service provider and customer. Constraints refer to the behaviour of services or resources, but of course they can only be tested by applying some *testing procedure* to the relevant monitoring data. The testing procedure will involve some mathematical manipulation to extract relevant aspects of the behaviour from the monitoring data.

III. SERSCIS METRICS

In SERSCIS, we aim to support metrics which will represent the base classes that capture the physical and mathematical nature of certain kinds of service behaviors and measurements. These are described below.

A. Absolute Time

This metric signifies when (what time and date) some event occurs. It can be measured simply by checking the time when the event is observed. Subclasses of this metric would be used to refer to particular events, e.g. the time at which a service is made available, the time it is withdrawn from service, etc. There are two types of constraints imposed on this metric. (1) a lower limit on the absolute time, encoding “not before” condition on the event. (2) an upper limit on the absolute, encoding a “deadline” by which an event should occur.

B. Elapsed Time

This metric just signifies how long it takes for some event to occur in response to some stimulus. It can be measured by recording the time when the stimulus arises, then checking the time when the subsequent event is observed and finding the difference. Subclasses of this metric would be used to refer to particular responses, e.g. the time taken to process and respond to each type of request supported by each type of service, or the time taken for some internal resourcing action such as the time for cleaners to reach an aircraft after it was scheduled and available. In the SERSCIS P-o-C, it should be possible to ask a consumer task for the elapsed times of all responses corresponding to the metric, and possibly to ask for the same thing in a wider context (e.g. from a service or service container). Constraints placed on elapsed time are (1) an upper limit on the elapsed time which encodes a lower limit on the performance of a service. (2) a lower limit which is typically used only in management policies to trigger actions to reduce the resource available if a service over-performs. If there are many events of the same type, one may wish to define a single constraint that applies to all the responses, so if any breaches the constraint the whole set is considered to do so. This allows one to test the constraint more efficiently by checking only the fastest and slowest response in the set. Sometimes it may be appropriate to define constraints that include more than one response time. For example, suppose a service supports aircraft refuelling but the amount of fuel supplied (and hence the time spent actually pumping fuel) is specified by the consumer – See Figure **2**.

 ISBN: 978-1-61208-109-0

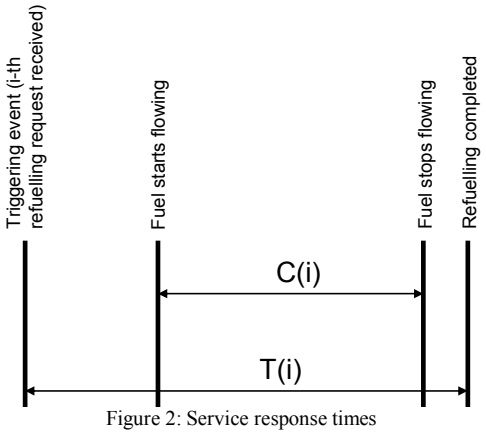

Figure 2: Service response times

In this situation the service provider can't guarantee the total response time T(i), because they have no control over the amount of time C(i) for which the fuel will actually flow into the aircraft. But they can control how long it takes for a fuel bowser to reach the aircraft after the refuelling request is received, and how long it takes to connect and disconnect the fuelling hoses and get clear after fuelling is completed, etc. So the service provider may prefer to specify a constraint on the difference between the two elapsed times. In SERSCIS, anything that is constrained should be a metric (to keep the SLA and policy constraint logic and schema simple), so in this situation one should define a new metric which might be called something like 'fuelling operation time'. One then has two options to *obtain its value (1) measure it directly so values are returned by the measurement procedure; or (2) define rules specifying the relationship between the new metric's value and the other metrics whose values are measured.*

C. Counter

This metric signifies how often events occurs since the start of measurement. It can be measured by observing all such events and adding one to the counter (which should be initialised to zero) each time an event occurs. In some situations it may be desirable to reset the counter to zero periodically (e.g. at the start of each day), so the metric can refer to the number of events since the start of the current period. In this case it may be appropriate to record the counter for each period before resetting it the retained value for the next period. Subclasses of this metric would be used to refer to particular types of events, e.g. the number of requests of each type supported by the service, or the number of exceptions, etc. In the SERSCIS P-o-C, it should be possible to ask a consumer task, service or container for the counters for each type of request and for exceptions arising from each type of request. Note that some types of request may only be relevant at the service or container level, and for these the counters will only be available at the appropriate level. Constraints here are upper and lower limits encoding the commitments not to send too many requests or generate too many exceptions or to trigger management actions. There are also limits on the ration between the numbers of events of different types.

D. Max and Min Elapsed Time

These metrics signify the slowest and fastest response to some stimulus in a set of responses of a given type, possibly in specified periods (e.g. per day). They can be measured by observing the elapsed times of all events and keeping track of the fastest and slowest responses in the set. Subclasses of this metric would be used to refer to particular types of response, e.g. times to process and respond to each type of service request, etc. In the SERSCIS P-o-C, it should be possible to ask a consumer task, service or container for the minimum and maximum elapsed times corresponding to the metric. Constraints on such metrics signify the range of elapsed times for a collection of responses. Only one type of constraint is commonly used: an upper limit on the maximum elapsed time, encoding a limit on the worst case performance of a service.

E. Mean Elapsed Time

This metrics signifies the average response to some stimulus for responses of a given type, possibly in specified periods. It can be measured by observing the elapsed times for all such responses, and keeping track of the number of responses and the sum of their elapsed times: the mean is this sum divided by the number of responses. Subclasses of this metric would be used to refer to particular types of response, e.g. times to process and respond to each type of service request, etc. In the SERSCIS P-o-C, it should be possible to ask a consumer task, service or container for the mean elapsed time corresponding to the metric. Constraints on this metric are the same as those for the elapsed time metric.

F. Elapsed Time Compliance

This metric captures the proportion of elapsed times for responses of a given type that don't exceed a specified time limit. Metrics of this type allow the distribution of elapsed times to be measured, by specifying one or more compliance metrics for different elapsed time limits (See Figure **3**).

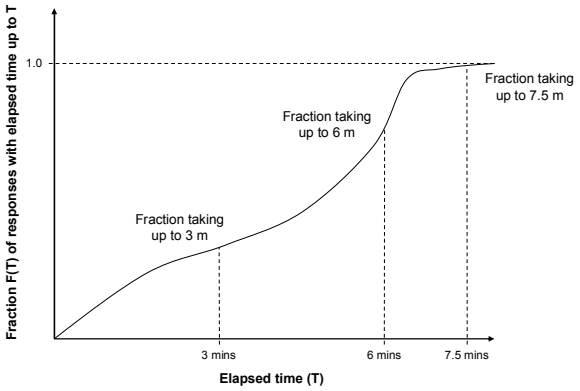

Figure 3: Elapsed time distribution

 ISBN: 978-1-61208-109-0

When measuring elapsed time compliance, it is convenient to make measurements for all the metrics associated with a distribution like Figure 3. One has to observe the elapsed times for all relevant responses, and keep track of the number of responses that were within each elapsed time limit, and also the total number of responses. The value of the elapsed time compliance metric at each limit is then the ratio between the number of responses that didn't exceed that limit and the total number of responses. Subclasses of this metric would be used to refer to particular types of responses and time limits. For example, one might define multiple elapsed time compliance metrics for different time limits for responses to each type of request supported by the service, and for some internal process time. In the SERSCIS P-o-C, it should be possible to ask a consumer task, service or container for the elapsed time compliance for responses corresponding to the metric. It may also be useful to support requests for all elapsed time compliance metrics for a given type of response, allowing the compliance of the entire distribution function to be obtained at once. Note that some types of request may only be relevant at the service or container level, and for these the elapsed time distribution function will only be available at the appropriate level. Constraints for this metric are normally expressed as lower (and sometimes upper) bounds on the value of the metric for specific responses and time limits. SLA commitments typically involve the use of lower bounds (e.g. 90% of responses within 10 mins, 99% within 15 mins, etc), but both upper and lower bounds may appear in management policies (e.g. if less than 95% of aircraft are cleaned within 10 mins, call for an extra cleaning team).

G. Non-recoverable resource usage and usage rate

These metrics capture the notion that services consume resources, which once consumed cannot be got back again (this is what we mean by non-recoverable). In most cases, non-recoverable usage is linked to how long a resource was used, times the intensity (or rate) of usage over that period. It can be measured by observing when a resource is used, and measuring either the rate of usage or the total amount of usage at each observation. Subclasses of the non-recoverable usage metric would be used to refer to the usage of particular types of resources, for example on CPU usage, communication channel usage, data storage usage etc. In the SERSCIS P-o-C, it should be possible to ask a consumer task, service or container for the usage rate at the last observation, and the total usage up to that point. Ideally this should trigger a new observation whose result will be included in the response. The response should include the absolute time of the last observation so it is clear whether how out of date the values in the response may be. Non-recoverable resource usage is characterized by functions of the form:

$$U(S,t) \geq 0 \quad (1)$$

$$\frac{dU(S,t)}{dt} \geq 0 \quad (2)$$

U represents the total usage of the non-recoverable resource by a set of activities S up to time t. The range of U is therefore all non-negative numbers, while the domain spans all possible sets of activities using the resource, over all times. In fact, U is zero for all times before the start of the first activity in S (whenever that may have been), and its time derivative is also zero for all times after the last activity has finished. The time derivative of U represents the rate of usage of the non-recoverable resource. This must be well-defined and non-negative, implying that U itself must be smooth (continuously differentiable) with respect to time, i.e. it can't have any instantaneous changes in value.

Constraints for non-recoverable usage and usage rate are typically simple bounds on their values. Both upper and lower bounds often appear in management policies to regulate actions to decrease as well as increase resources depending on the load on the service:

$$L_0 \leq U(S,t_0) - U(S,t_0) \leq L_1 \quad (3)$$

represents a constraint on the minimum and maximum total usage for a collection of activities S in a time period from t_0 to t_1, while:

$$M_0 \leq \frac{dU(S,t)}{dt} \leq M_1, \forall t: t_0 \leq t \leq t_1 \quad (4)$$

represents a constraint on the maximum and minimum total usage rate for a collection of activities S during a time period from t_0 to t_1. Note that it is possible to have a rate constraint (4) that allows a relatively high usage rate, in combination with a total usage constraint (3) that enforces a much lower average usage rate over some period. Alternatively, a contention ration could be introduced for usage rate constraints to handle cases where a resource is shared between multiple users but may support a high usage rate if used by only one at a time.

H. Maximum and Minimum Usage Rate

These metrics capture the range of variation in the usage rate (possibly in specified periods, which is described above. They can be measured by simply retaining the maximum and minimum values of the usage rate whenever it is observed by the measurement procedure. Subclasses of these metrics would be used to refer to maximum and minimum usage for particular types of resources. Constraints on maximum and minimum usage rate take the form of simple bounds on their values. Note that if we constrain maximum usage rate to be up to some limit, and the usage rate ever breaches that limit, then the constraint is violated however the usage rate changes later.

 ISBN: 978-1-61208-109-0

I. State

This metric captures the current state of a service, with reference to a (usually finite) state model of the service's internal situation (e.g. the value of stored data, the status of supplier resources, etc). The value of the metric at any time must be a state within a well-defined state model of the service, usually represented as a string signifying that state and no other. It can be measured by observing the internal situation of the service and mapping this to the relevant state from the state model. In the SERSCIS PoC implementation, it should be possible to ask a task, service or container for its current state. Note that the state model of a service will normally be different from the state model of tasks provided by the service, and different from the state model of the container providing the service. State is an instantaneous metric – a measurement of state gives the state at the time of observation only. To obtain a measure of the history of state changes one should use state occupancy metrics or possibly non-recoverable usage metrics for each possible state of the service. Subclasses of the state metric will be needed to refer to particular state models and/or services. Constraints can be used to specify which state a service should be in, or (if the state model includes an ordering of states, e.g. security alert levels), what range of states are acceptable.

J. State Occupancy

This metric captures the amount of time spent by a task in a particular state (possibly in specified periods). It can be measured by observing state transitions and keeping track of the amount of time spent in each state between transitions. Note that for this to be practical one must predefine a state model for the task encompassing all its possible states, in which the first transition is to enter an initial state when the task is created.

The state of a resource on a service is a function of time:

$$S_i(t) \in \Sigma, \forall t \geq t_0 \qquad (5)$$

where $S_i(t)$ is the state of resource i at time t, $\sum$ is the set of possible states (from the resource state model) and t_0 is the time resource i was created. Constraints on state occupancy are bounds on the proportion of time spent in a particular state, or the ratio between the time spent in one state and time spent in one or more other states.

K. Data Accuracy

This metric captures the amount of error in (numerical) data supplied to or from a service, compared with a reference value from the thing the data is supposed to describe. The two main aspects of interest with this particular metric are the precision of the data (how close to the reference value is the data supposed to be) and the accuracy of the data (how close to the reference value the data is, compared to how close it was supposed to be). Subclasses of data accuracy may be needed to distinguish between different types of data used to describe the thing of interest (single values, arrays etc), and different ways of specifying precision (precision in terms of standard deviation, confidence limit etc), as well as to distinguish between things described by the data (e.g. aircraft landing times, fuel levels or prices). In the SERSCIS P-o-C, we are only really interested in the accuracy of predictions for the absolute time of future events, including the point when an aircraft will be available so turnaround can start (an input to the ground handler), the point when the aircraft will be ready to leave, and various milestones between these two points (e.g. the start and end of aircraft cleaning, etc). Constraints on accuracy are typically just upper bounds on the accuracy measure, e.g. accuracy should be less than 2.0. Such constraints apply individually to each data value relating to a given reference value.

L. Data Precision

This is a simple metric associated with the precision bands for data supplied to or from a service. Data that describes some reference value should always come with a specified precision, so measuring the precision is easy – one just has to check the precision as specified by whoever supplied the data. The reason it is useful to associate a metric with this is so one can specify constraints on data precision in SLA, to prevent data suppliers evading accuracy commitments by supplying data very poor (wide) precision bands. Subclasses of data precision are typically needed for different kinds of things described by data, and different sources of that data. For example, one might define different metrics to describe the precision in scheduled arrival times (taken from an airline timetable) and predicted arrival times (supplied by Air Traffic Control when the aircraft is en-route). Note that precision (unlike accuracy) is not a dimensionless number – it has the same units as the data it refers to, so metric subclasses should specify this. In the SERSCIS P-o-C testbed, it should be possible to ask a consumer task for the precision of data supplied to or by it. The response should ideally give the best, worst and latest precision estimates for the data corresponding to the metric. Constraints on data precision are simple bounds on its value. Typically they will appear in SLA, and define the worst-case precision that is acceptable to both parties. If data is provided with worse precision than this, the constraint is breached. This type of constraint is normally used as a conditional clause in compound constraint for data accuracy or accuracy distribution.

M. Data Error

This is a simple metric associated with the error in a data item relative to the reference value to which it relates. In some situations we may wish to specify and measure commitments for this 'raw' measure of accuracy, independently of its supposed precision. Subclasses of data error are typically needed for different kinds of things described by data, and different sources of data. In the SERSCIS P-o-C testbed, it should be possible to ask a consumer task for the error in data supplied to or by it once the reference value is known to the service. The response should ideally give the best, worst and latest error for data

 ISBN: 978-1-61208-109-0

sent/received corresponding to the metric. Constraints on data error are simple bounds on its value. Typically they will appear in SLA, and define the worst-case error that is acceptable to both parties. If data is provided and turns out to have an error worse than this, the constraint is breached.

N. Data Accuracy Compliance

This metric captures the proportion of data items in a data set provided to or from a service whose accuracy is not worse than a specified limit. This metric is mathematically similar to the elapsed time compliance metric, and as before we may wish to use several accuracy compliance metrics for the same data at different accuracy levels, to approximate a data accuracy distribution function. Accuracy compliance can be measured by keeping track of the total number of data items, and how many of these had accuracy up to each specified level. The value of the metric is then the fraction of data items whose accuracy is within the specified level. In the SERSCIS P-o-C testbed, subclasses of accuracy compliance are typically used to distinguish between different accuracy levels, types of data and methods for defining precision, for data forecasting the time of events. To construct accuracy distributions it is necessary to classify those events so we know which forecasts to include in each distribution function. It should be possible to ask consumer tasks, services or service containers for the value of these compliance metrics. Constraints on accuracy compliance just specify bounds on the metric, thus specifying what proportion of data items can have accuracy worse than the corresponding accuracy limit.

O. Auditable Properties

Auditable property metrics are used to express whether a service satisfies some criterion that can't be measured, but can only be verified through an audit of the service implementation and behaviour. An auditable property will normally be asserted by the service provider, who may also provide proof in the form of accreditation based on previous audits in which this property was independently verified. Auditable properties are usually represented as State metrics: a state model is devised in which the desired property is associated with one or more states, which are related (out of band) to some audit and if necessary accreditation process. Subclasses are used to indicate different auditable properties and state models. Auditable property constraints typically denote restrictions on the resources (i.e. supplier services) used to provide the service. For example, they may specify that only in-house resources will be used, that staff will be security vetted, or that data backups will be held off site, etc. In SERSCIS, such terms are also referred to as Quality of Resourcing (QoR) terms. As with other state-based descriptions, auditable properties may be binary (true or false), or they may be ordered (e.g. to describe staff with different security clearance levels). It is also possible to treat Data Precision (and other data characteristics) as an auditable property which does not correspond to a state model.

IV. RELATED WORK

Characterizing the performance of adaptive real-time systems is very difficult because it is difficult to predict the exact run-time workload of such systems. Transient and steady state behavior metrics of adaptive systems were initially drafted in [4], where the performance of an adaptive was evaluated by its response to a single variation in the application behavior that increased the risk of violating a performance requirement. A very simple set of metrics are used: *reaction time* which is the time difference between a critical variation and the compensating resource allocation, *recovery time* by which system performance returns to an acceptable level, and performance laxity which is the difference between the expected and actual performance after the system returns to a steady state. These metrics are further specialized in [1] by the introduction of *load profiles* to characterize the types of variation considered including *step-load* (instant) and *ramp-load* (linear) changes, and a *miss-ratio* metric which is the fraction of tasks submitted in a time window for which the system missed a completion deadline. System performance is characterized by a set of miss-ratio profiles with respect to transient and steady state profiles. A system is said to be stable in response to a load profile if the system output converges as the time goes to infinity, while transient profiles can measure responsiveness and efficiency when reacting to changes in run-time conditions. The SERSCIS-Ont metrics provide a superset of these concepts, appropriate to a wider range of situations where accuracy and reliability may be as important as performance and stability.

A more recent alternative approach to defining adaptive system metrics is given by [6,7]. Here the focus is on the system engineering concerns for adaptivity, and metrics are categorized into four types: *architectural* metrics which deal with the separation of concerns and architectural growth for adaptive systems [2], *structural* metrics which provide information about the role of adaptation in the overall functionality of a system (and vice versa), *interaction* metrics which measure the changes in user interactions imposed by adaptation, and *performance* metrics which deal with the impact of adaptation on system performance, such as its response time, performance latency, etc [2]. The focus of SERSCIS-Ont is to provide concrete and mathematically precise metrics covering performance and some aspects of interactivity, which can be used in such a wider engineering framework.

The most closely related work is found in the WSMO initiative [3], which has also formalized metrics for resource dependability. This was done with the intention of providing QoS aware service oriented infrastructures. Semantic SLA modeling using WSMO focuses principally on automated service mediation and on the service execution infrastructure [3]. By adding semantic descriptions for

 ISBN: 978-1-61208-109-0

service parameters it is possible for agents to discover and rank services automatically by applying semantic reasoning. The WSMO initiative focused its modeling efforts on capturing service consumer requirements, which can then be used for service discovery. Work in [5] extends the WSMO ontology to include QoS and non-functional properties. This includes providing formal specifications for service level agreements including the units for measurement, price, CPU usage etc. However, the focus is still to support the description of services for orchestration purposes (service discovery and selection). SERSCIS-Ont is more even-handed. It can be used for service discovery and selection, but it is also designed to support service operators by introducing service protection measures from a provider's perspective such as the usage limits, service access and control decisions, as well as workflow adaption, etc.

SERSCIS-Ont is thus also related to the development and service management specifications such as WSDM. The WSDM-MOWS specification [9] defines 10 metrics which are used to measure the use and performance of a general Web Service. These include NumberOfRequests, NumberOfFailedRequests and NumberOfSucessfulRequests which count the messages received by the Web Service end point, and whether the service handles them successfully. In SERSCIS-Ont we have a more general Counter metric, of which these WSDM-MOWS metrics can be regarded as subclasses specifically for Web Service management. WSDM-MOWS also defines ServiceTime (the time taken by the Web Service to process all its requests), and MaxResponseTime and LatestResponseTime. In SERSCIS-Ont these would be modeled as subclasses of usage and elapsed time, and SERSCIS-Ont then provides additional metrics such as min/max/mean responses and response time compliance metrics. WSDM-MOWS specifies a state model for Web Service operation with states {UpState, DownState, IdleState, BusyState, StoppedState, CrashedState, SaturatedState}, and metrics CurrentOperationalState and LastOperationStateTransition all of which can be handled easily by SERSCIS-Ont. The one area where WSDM-MOWS goes beyond SERSCIS-Ont is in providing metrics for the size of Web Service request and response messages: MaxRequestSize, LastRequestSize and MaxResponseSize. These can be modeled with difficulty using SERSCIS-Ont usage metrics, but if SERSCIS-Ont were applied to Web Service management, some extensions would be desirable.

V. VALIDATION EXPERIMENTS

To verify that SERSCIS-Ont really is applicable to the management of service performance and dependability, the project is conducting two types of experiments. Testbeds are being developed comprising SERSCIS dependability management tools along with emulated application services based on air-side operations at Vienna Airport. This will be a discrete event simulation in which realistic application-level requests and responses are produced, and the full (not emulated) management tools will be tested using SERSCIS-Ont metrics in service level agreements and monitoring and management policies.

Until the testbed is ready, SERSCIS validation work has focused on the use of stochastic process simulation based on queuing theory [10]. A simplified Markov chain model was developed for a single aircraft refueling service, and the resulting equations solved numerically to compute the expected behavior. This approach is faster and easier to interpret than a discrete event simulation, though it uses simpler and less realistic models of services and their interactions.

The basic model of the refueling service assumes that around 20 aircraft arrive per hour and need to be refueled. The service provider has 3 bowsers (fuel tankers) which can supply fuel to aircraft at a certain rate. The time taken for refueling varies randomly between aircraft depending on their needs and how much fuel they still have on landing, but the average time is 7.5 minutes. However, with only 3 bowsers, aircraft may have to wait until one becomes available before refueling can start. The SERSCIS-Ont metrics used to describe this service are:

- a counter metric for the number of aircraft refueled, and an associated usage rate metric for the number of aircraft refueled per hour;
- a non-recoverable usage rate metric for the time the bowsers spend actually refueling aircraft, from which we can also obtain the resource utilization percentage;
- an elapsed time metric for the amount of time spent by aircraft waiting for a bowser (the refueling service can't control how long the refueling takes, so QoS is defined in terms of the waiting time only); and
- elapsed time compliance metrics for the proportion of aircraft that have to wait for different lengths of time between 0 and 20 minutes.

We also assume that the service will refuse an aircraft, i.e. tell it to use another refueling company rather than wait, if it would become the 10^{th} aircraft in the queue. This is captured by a further counter metric, which is used to find the proportion of arriving aircraft that are refused service.

The first simulation considered an unmanaged service (no SLAs), and produced the following behavior (See Table 1):

TABLE 1: UNMANAGED SERVICE SIMULATION

Metric	Value
Service load	20 aircraft / hour
Service throughput	19.5 aircraft / hour
Percentage of aircraft that don't have to wait	33.6%
Percentage that don't have to wait more than 10 mins	74.6%
Percentage that don't have to wait more than 20 mins	94.4%
Percentage of aircraft refused service	2.6%
Mean waiting time	6.1 mins
Resource utilization	81.2%

 ISBN: 978-1-61208-109-0

The QoS is relatively poor because the random variation in aircraft arrival and refueling times means queues can build up, leading to a high proportion of aircraft having to wait, and some having to wait for a long time or even being sent to other service providers.

To investigate how the metrics could be used to manage the service, the simulation was extended so airlines must have an SLA with the service provider before they can use the service. Each SLA lasts on average 1 week, and allows an airline to refuel an average of 3 aircraft per hour. The extended model assumed about one new SLA per day would be signed, giving an average load roughly similar to the total load in the first simulation. We also assumed the service provider would refuse to agree more than 12 SLA at a time, so the load could temporarily rise up to 50% higher than the capacity of its resources. We wished to investigate how well the use of SLA as a pre-requisite for service access allowed such overloads to be managed. The results of this second simulation were as follows (See Table 2):

TABLE 2: MANAGED SERVICE SIMULATION

Metric	Value
Service load	0-36 aircraft / hour
Service throughput	21.1 aircraft / hour
Percentage of aircraft that don't have to wait	22.4%
Percentage that don't have to wait more than 10 mins	60.4%
Percentage that don't have to wait more than 20 mins	89.7%
Percentage of aircraft refused service	4.9%
Mean waiting time	9.4 mins
Resource utilization	87.8%

While the use of this SLA allowed the service provider to anticipate the load from a pool of potential consumers, it couldn't improve QoS with a fixed set of resources. In fact, the compliance metrics are now much worse than before, with only a small increase in the total throughput because the load exceeds the resource capacity around 25% of the time. Further tests showed that reducing the number of SLA the service accepts doesn't help much as this only lowers the long term average load, whereas overloads and long queues arise from shorter-term fluctuations. The limit would have to be much lower (and the throughput substantially lower) before the compliance metrics were good enough to be of interest to customers.

The final experiment used a different type of SLA in which each customer can still have 3 aircraft serviced per hour on average, but only one at a time. To handle this, we used a non-recoverable usage rate metric for the number of aircraft in the system and specified in the SLA that this could not exceed 1. This simulation produced the following (See Table 3):

TABLE 3: CONSTRAINED SLA SERVICE SIMULATION

Metric	Value
Service load	0-36 aircraft / hour
Service throughput	17.9 aircraft / hour
Percentage of aircraft that don't have to wait	50.6%
Percentage that don't have to wait more than 10 mins	96.0%
Percentage that don't have to wait more than 20 mins	99.9%
Percentage of aircraft refused service	0%
Mean waiting time	3.4 mins
Resource utilization	74.7%

Evidently, if this last type of SLA were enforced by a suitable management procedure, it would allow the service to protect itself from overloads, without a huge drop in the service throughput. Further experiments showed that if the permitted long-term load per SLA were pushed up to 3.5 aircraft per hour, the throughput would reach 19.7 aircraft per hour (more than the original unmanaged service), yet the compliance metrics would stay above 90%. This provides a good indication that the SERSCIS-Ont metrics can be used to describe service management and protection constraints, as well as consumer QoS measurements and guarantees.

VI. CONCLUSIONS

This paper describes a base metric model that provides a uniform abstraction for describing service behavior in an adaptive environment. Such an abstraction allows services to be composed into value chains, in which consumers and providers understand and can manage their use of services according to these metrics.

A service provider, having analyzed the application service that it is offering, defines a metric ontology to describe measurements of the relevant service behavior. This ontology should refer to the SERSCIS base ontology, and provide subclasses of the base metrics to describe each relevant aspect of service behavior. Note that while each service provider can in principle define their own metrics ontology, it is may be advantageous to establish 'standard' ontologies in particular domains – this reduces the need for translation of reported QoS as it crosses organizational boundaries.

Validation simulations provide a good indication that the SERSCIS-Ont metrics are useful for describing both service management and protection constraints, and service dependability and QoS guarantees.

 ISBN: 978-1-61208-109-0

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement no. 225336, SERSCIS.

REFERENCES

[1] C. Lu, J.A. Stankovic, T.F. Abdelzaher,G.Tao, S.H. Son and M.Marley, "Performance Specifications and Metrics for Adaptive Real-Time Systems,"In Real-Time Systems Symposium 2000.

[2] C. Raibulet and L. Masciadri. "Evaluation of Dynamic Adaptivity Through Metrics: an Achievable Target?". In the paper proceedings of the 8th working IEEE/IFIP Conference on Software Architecture. WICSA 2009.

[3] D.Roman, U.Keller, H. Lausen, R.L.J. de Bruijn, M. Stolberg, A.Polleres, C.Feier, C.Bussler and D.Fensel. "Web service modelling ontology". Applied Ontology. I (1):77-106, 2005.

[4] D. Rosu, K. Schwan, S. Yalamanchili and R. Jha, "On Adaptive Resource Allocation for Complex Real-Time Applications," 18th IEEE Real-Time Systems Symposium, Dec., 1997. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[5] I. Toma, D. Foxvog, and M.C. Jaeger. "Modelling QoS characterists in WSMO". In: Proceedings of the 1st workshop on Middleware for Service Oriented Computing. November 27-December 01, 2006.

[6] L. Masciadri, "A Design and Evaluation Framework for Adaptive Systems", MsC Thesis, University of Milano-Bicocca, Italy, 2009.

[7] L. Masciadri, and C. Raibulet, "Frameworks for the Development of Adaptive Systems: Evaluation of Their Adaptability Feature Software Metrics", Proceedings of the 4th International Conference on Software Engineering Advances, 2009, in press.

[8] Collected Works of John von Neumann, 6 Volumes. Pergamon Press, 1963.

[9] WSDM-MOWS Specificaiton. www: http://docs.oasis-open.org/wsdm/wsdm-mows-1.1-spec-os-01.htm (Last accessed Aug 2010).

[10] D. Gross and C.M. Harris. Fundamentals of Queueing Theory. Wiley, 1998.