SERSCIS-Ont

Evaluation of a Formal Metric Model using Airport Collaborative Decision Making

Mike Surridge, Ajay Chakravarthy,
Maxim Bashevoy, Joel Wright, Martin Hall-May
IT Innovation Centre
University of Southampton
Southampton, UK
{ms,ajc,mvb,jjw,mhm}@it-innovation.soton.ac.uk

Roman Nossal
Austro Control
Österreichische Gesellschaft für Zivilluftfahrt mbH
Vienna, Austria
roman.nossal@austrocontrol.at

***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. The approach is evaluated using a case study based on an airport-based collaborative decision-making scenario. Two experimental approaches are presented: the first based on stochastic process simulation, the second on discrete event-based simulation.**

Keywords-adaptive metrics; SOA; measurements; constraints; QoS; discrete event simulation.

I. INTRODUCTION

This paper presents the SERSCIS-Ont metric ontology first introduced in [11], together with an expanded evaluation section.

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 (PoC) 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 describes the scenario and experiments that are used to test the applicability of the SERSCIS metrics. Section VI presents the results of the validation experiment carried out using stochastic process modelling and simulation. Sections VIII and IX elaborate the experimental scenario by describing, respectively, Key Performance Indicators (KPI) for each actor and failure scenarios. These are then demonstrated in Section IX using the results of a discrete event-based simulation experiment. Finally, we conclude the paper in Section X.

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.

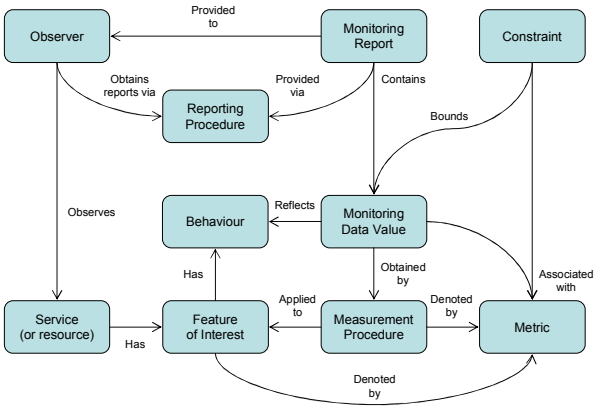

Figure 1. Metrics, Measurements and Constraints

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 behaviours 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 PoC, 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.

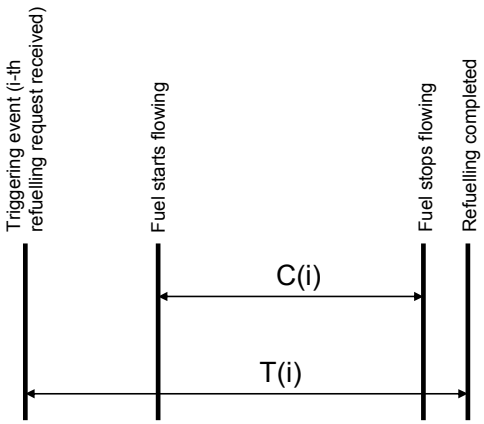

Figure 2. Service response times

In this situation, the service provider cannot 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 PoC, 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 PoC, 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 PoC, 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 do not 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.).

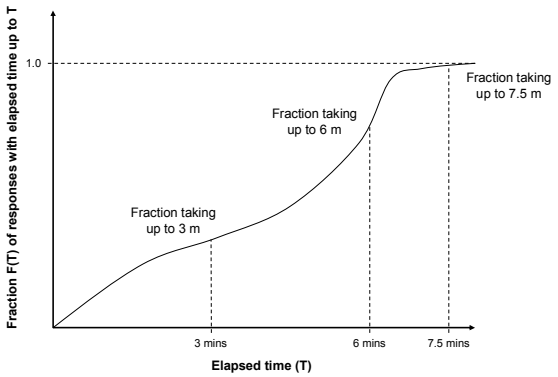

Figure 3. Elapsed time distribution

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 did not 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 PoC, 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 PoC, 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 cannot 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:

$$L_0 \leq U(S,t_0) - U(S,t_1) \leq L_1 \quad (3)$$

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.

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:

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

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 PoC, 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 PoC 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 PoC 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 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 PoC 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 metrics; 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 cannot 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 behaviour 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 behaviour 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 modelling using WSMO focuses principally on automated service mediation and on the service execution infrastructure [3]. By adding semantic descriptions for service parameters it is possible for agents to discover and rank services automatically by applying semantic reasoning. The WSMO initiative focused its modelling 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 modelled 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 modelled 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: the first involving stochastic process simulation and the second extends to discrete event simulation. In the latter case, a testbed has been developed which comprises SERSCIS dependability management tools along with emulated application services based on air-side operations at Vienna Airport. This is a discrete event simulation in which realistic application-level requests and responses are produced, and the full (not emulated) management tools are tested using SERSCIS-Ont metrics in service level agreements and monitoring and management policies.

A. Scenario Description

The scenario used to validate the metric model is based on Airport Collaborative Decision Making (A-CDM). A-CDM is an approach to optimizing resource usage and improving timeliness at an airport. It is about all partners at an airport working together, openly sharing accurate information and – based on the information – making decisions together. Through the use of A-CDM predictability of airport operations is improved. All actions involved in turning around an aircraft can be planned more accurately and the plans can more easily be controlled with respect to the actual operation.

A-CDM also has a European, network-wide perspective. The Central Flow Management Unit (CFMU) of Eurocontrol monitors the capacity of airspace sectors and imposes restrictions by issuing so-called slots in case congestion might arise. Currently, this planning is mainly based on flight plan information that is filed up to three hours before the actual flight. Changes, in particular last minute changes e.g., due to late passengers, are not taken into account. Hence, everyday a huge amount of airspace capacity is wasted due to inaccurate information. The Airports applying A-CDM can more accurately determine the take-off time of departing flights. CFMU can then update their network planning based on information that closely reflects the real traffic to be expected. Hence, slot wastage is minimized for the benefit of all airspace users.

The testbed scenario for the evaluation is based on the workflow that is executed during an aircraft turn-around. The workflow represents the interaction of the main actors in a turn-around, i.e., the ANSP, a ground handler and ramp service providers. Each step in the workflow uses a service to perform the step. Services are provided by different actors. Most services can be provided by more than one service provider. In this case the service user has the choice of the service provider, for which he has to take into account several Quality-of-Service criteria.

The workflow, which is shown in Figure 4. consists of three sub-workflows being executed in parallel. After the aircraft goes in-block passenger disembarkation starts. At the same time a baggage handler starts to offload the luggage from the aircraft. The third sub-workflow deals with refuelling the aircraft. It can only be started after disembarkation of passengers is finished. Going back to the first sub-workflow, when disembarkation is finished an optional security check of the plane for left items can be performed by either the crew or a security company under the crew's supervision. When this is done the crew leaves the aircraft. Cleaning of the aircraft and catering commence in parallel. For the latter to be released the new crew is required as they have to check the number of meals provided. Upon completion of cleaning and release of catering another optional security check can be performed given that refuelling has completed as well. After the security check embarkation of passengers can begin if the landside workflow is ready for boarding.

The second sub-workflow is concerned with offloading the luggage and loading the new luggage. It is completely independent of the passenger and cabin-related workflow. Finally, the third sub-workflow has the purpose of refuelling the aircraft. As mentioned above, it can only commence once disembarkation has completed. In turn, completion of refuelling is a precondition for passenger embarkation.

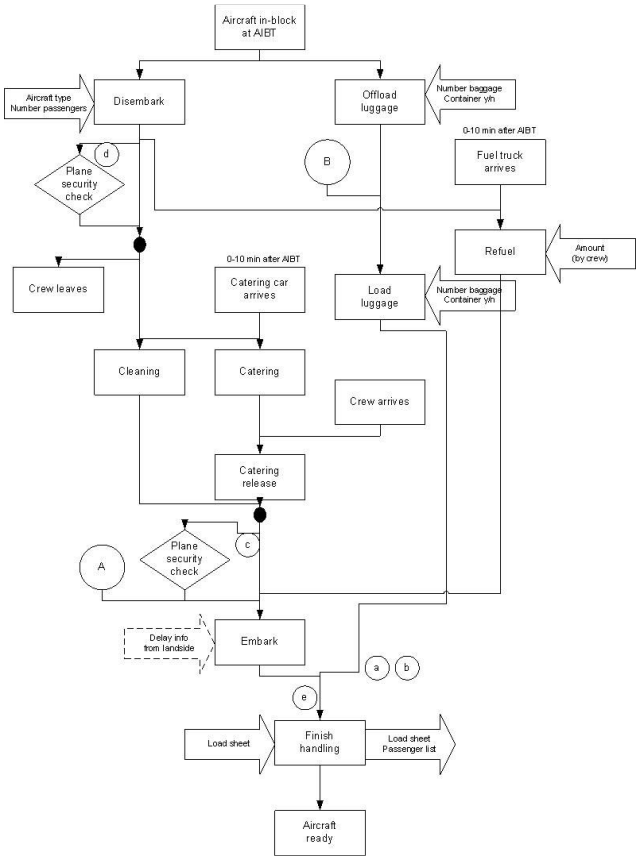

Figure 4. Airside Workflow for Aircraft Turnaround

B. Validation Objectives

This evaluation is done by validation, which applies the system to a chosen scenario. From the results of validation one can derive the usefulness of the system for the given application.

In the chosen A-CDM scenario the SERSCIS mechanisms and tools must demonstrate two characteristics:

- They must be able to implement and execute the real-world scenario in the fault-free case. This property ensures that the SERSCIS tools capture the scenario requirements and are able to accompany the execution of the processes. Note that in order to gain acceptance with the potential users, the tools and mechanisms must adapt to the real-world processes, not the other way round.
- They must be able to handle failure cases and improve the execution of the processes in these cases. While the above fault-free case shows the possibility to execute the processes with SERSCIS support, this validation aims at proving the added value of SERSCIS. The tools and mechanisms must improve the handling of failure cases.

Several test runs were conducted to demonstrate the above characteristics. The properties of the SERSCIS tools and mechanisms were evaluated by use of so-called Key Performance Indicators (KPIs), which are described in Section VII.

VI. Stochastic Process Simulation Experiments

SERSCIS validation work initially focused on the use of stochastic process simulation based on queuing theory [10]. A simplified Markov chain model was developed for a single aircraft refuelling service, and the resulting equations solved numerically to compute the expected behaviour. 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 refuelling service assumes that around 20 aircraft arrive per hour and need to be refuelled. The service provider has 3 bowsers (fuel tankers), which can supply fuel to aircraft at a certain rate. The time taken for refuelling 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 refuelling can start. The SERSCIS-Ont metrics used to describe this service are:

- a counter metric for the number of aircraft refuelled, and an associated usage rate metric for the number of aircraft refuelled per hour;
- a non-recoverable usage rate metric for the time the bowsers spend actually refuelling 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 refuelling service cannot control how long the refuelling 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 refuelling company rather than wait, if it would become the 10th 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 behaviour (See Table I):

TABLE I. Unmanaged Service Simulation

Metric	Value
Service load	20 aircraft / hour
Service throughput	19.5 aircraft / hour
Percentage of aircraft that do not have to wait	33.6%
Percentage that do not have to wait more than 10 mins	74.6%
Percentage that do not 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%

The QoS is relatively poor because the random variation in aircraft arrival and refuelling 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 II):

TABLE II. MANAGED SERVICE SIMULATION

Metric	Value
Service load	0-36 aircraft / hour
Service throughput	21.1 aircraft / hour
Percentage of aircraft that do not have to wait	22.4%
Percentage that do not have to wait more than 10 mins	60.4%
Percentage that do not 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 could not 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 does not 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 III. CONSTRAINED SLA SERVICE SIMULATION

Metric	Value
Service load	0-36 aircraft / hour
Service throughput	17.9 aircraft / hour
Percentage of aircraft that do not have to wait	50.6%
Percentage that do not have to wait more than 10 mins	96.0%
Percentage that do not 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.

VII. BUSINESS-LEVEL OBJECTIVES AND KEY PERFORMANCE INDICATORS

While the above-mentioned description set the framework for the scenario, the identification of failure and threat scenarios requires a more in–depth look. In order to determine relevant service disruptions two additional pieces of information are required:

- A definition of the business-level objectives for each player in the scenario.
- An identification of the Key Performance Indicators (KPIs) used to measure the objective achievement.

Starting from the top-level CDM system business-level objectives are identified for each stakeholder in the scenario. These describe why the stakeholders participate in the CDM system and what they want to achieve. Each stakeholder's objectives determine the individual goals as well as the contribution to the higher-level goals of CDM overall.

A similar picture is drawn for the KPIs. At each level and stakeholder the KPIs should be usable to measure the achievement of the business-level objectives. At the same time they are grouped according to their contribution to the higher-level KPIs. The use of KPI enables SERSCIS to focus on system behaviour that is directly related to business performance, both of the A-CDM cell (i.e., the system as a whole), and of the individual stakeholders that contribute to it. This helps to ensure that SERSCIS only takes action when a problem really is a problem.

See Figure 5. for the concrete business-level objectives and KPIs for the Airport CDM scenario.

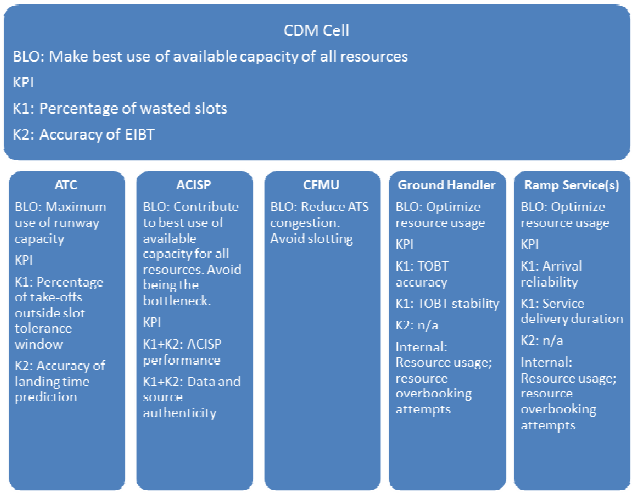

Figure 5. Business-level Objectives and Key Performance Indicators

The following sections describe the objectives and KPIs for the individual stakeholders. Please note that these sections only list KPIs that are of relevance to the overall A-CDM objectives and KPIs. Naturally there are several additional KPIs for each stakeholder, which they may use to assess their own performance. It was considered sufficient to use a few individual stakeholder KPIs in this evaluation. This ensures that SERSCIS can handle situations where individual and community goals may differ, but without needing to emulate the individual actors in excessive detail.

A. CDM Cell

The objective at this highest level is to make optimal use of the available resources. Note that this does not mean to increase any capacity, but to increase the usage of existing capacity.

The achievement level of this goal is measured by two KPIs.

1) Percentage of wasted slots (slots allocated but not used) (K1)

This will be measured on a monthly basis by measuring the wasted slots and dividing this figure by the number of totally allocated slots. Total allocated slots is given by the number of CTOTs issued by CFMU for flights departing from the airport. Wasted slots are defined as allocated slots (CTOTs) that passed without the aircraft departure or allocated slots (CTOTs) that have been changed within 15 minutes before the CTOT[1].

This KPI indirectly includes external requirements from the CFMU. It is the objective of the CFMU to reduce congestions and to reduce the number of wasted slots.

2) Accuracy of EIBT (K2)

This parameter is the basis for optimal resource scheduling and dispatching for ground handling and ramp services. The EIBT is the estimated time when the aircraft goes in-block at the stand. Hence this is the time when ground handling should start.

Measurement is done by mean square deviation between EIBT and AIBT for all flights of one day. EIBT is taken at FIR entry and at commencement of final approach for measurement purposes. The suggested goal is to achieve an accuracy of +/- 3 minutes at FIR entry and +/-1 minute on final approach.

EIBT accuracy is determined by:

- The accuracy of the landing time prediction (ELDT) and the updates to this and
- The accuracy of the taxi time prediction (EXIT).

While the latter factor originates from within the CDM cell (being provided by the ACISP), the first is provided either by the CDM actor ATC or externally by CFMU. Neither this input factor not EIBT accuracy itself are emulated in the current (proof of concept) testbed. The KPI is therefore listed here for completeness purpose only. The testbed and its evaluation at this stage focused on K1.

B. Central Flow Management Unit

The objective for this unit is to reduce congestions in the European air-traffic system and to avoid slotting[2] wherever possible.

The CFMU is not further detailed as it only acts as a value provider in the simulation. Beyond this CFMU's real functionality is not simulated.

C. A-CDM Information Sharing Platform

The objective for the ACISP is to deliver a performance that allows all stakeholders and the entire A-CDM system to achieve their goals. This performance goal also includes certain quality criteria with regard to data handling, in particular data consistency and data accuracy.

This is measured by the ACISP performance, i.e., the delay in forwarding values the CISP has received. This is simply measured by the sum of differences between reception and according sending time of a value divided by the number of such forwarding operations. This KPI also contributes to K1 and K2.

The ACISP as central data repository must meet high security requirements. Some of the data it stores are sensitive with respect to competition; others might influence physical security if exposed to the wrong person or if data from a wrong source are incorporated. It is also important that the data is accessible to those who have a right to use it.

To deal with this, a wide variety of metrics can be used, including:

- the accuracy of data retrieved by another actor: inaccuracy may indicate it is forged or corrupted (insecure update), in the absence of other explanations; or

[1] This assumes that a slot changed within the last 15 minutes before CTOT cannot be re-used for another flight by CFMU.

[2] "Slotting" is the process of issuing departure slots for flights if the calculation by CFMU shows a potential congestion anywhere in the enroute part. In other words, if the combined flight trajectories of all flights result in a capacity demand exceeding the capacity of any sector along the route, slots are issued for all flight passing this sector.

- the timeliness of data retrieved by another actor: if data updates are not available soon after they are made, or in the worst case, not available until after they are needed, this may indicate an availability problem.

Data confidentiality is difficult to monitor, as it is impossible to prove the null hypothesis that the data has NOT been accessed by an authorised party. One could seek to measure the number of known confidentiality breaches, which may be an indicator (albeit imperfect) of the number of actual breaches. A more common option is to ensure the data service has access control in place and to check the integrity of its implementation, e.g., through the use of vetted staff and accredited software.

Access control to ACISP data is implemented in the PoC testbed, but confidentiality breaches are not simulated yet, as they cannot directly cause a degradation of the infrastructure. However, data accuracy and timeliness are measures that can be used in the PoC evaluation.

D. Air Traffic Control

For ATC representing the Air Navigation Service Provider (ANSP) the business level is to maximize runway capacity and to reduce congestion of the European air traffic system while at the same time limiting or reducing the air-traffic controller workload. The second goal is not A-CDM specific and will not be regarded any further here.

For measuring the first objective two KPIs are devised. The first KPI helps to ensure that the European air traffic system makes best use of the available capacity by measuring the percentage of take-offs outside the so called slot tolerance window (STW, -5/+10 minutes around the Calculated Take-Off Time CTOT). This is performed by counting take-offs outside the STW and dividing this by the number of take-offs of regulated flights, i.e., flights that have a CTOT assigned. This directly contributes to K1.

Secondly the accuracy of the landing time prediction is measured, which reflects the ATC contribution to turn-around optimisation. For this the ELDT is compared to the ALDT. The concrete measurement is done by calculating the mean square deviation between ELDT and ALDT for all flights of one day. ELDT is taken at FIR entry and at commencement of final approach for measurement purposes. The suggested goal is to achieve an accuracy of +/- 3 minutes at FIR entry and +/-1 minute on final approach. In the PoC testbed, the ATC is not represented by an explicit service emulator, so any error in the landing time prediction forms part of the simulation input. For this reason, it is not used here, as already explained in section VII.A.

E. Ground Handler

The ground handler strives to optimize resource usage of his own and indirectly of the ramp services' resources. This involves human resources as well as equipment.

For the evaluation of this business level objective two internal KPIs are devised, which do not contribute to K1 nor to K2. They solely reflect the resource usage. Indicator 1 averages the usage of a type of resource over the period of a day, where usage is defined as percentage of resources occupied in comparison to resources available. One indicator is required for each type of resource.

The second indicator aims at avoiding overbooking. Per type of resource the number of occasions during a day are counted, when the service consumer tries to obtain resources beyond the available. Again, one indicator is required for each type of resource.

With respect to the overall A-CDM goals the ground handler contributes by accurately predicting the aircraft's TOBT. This is evaluated by two KPIs, TOBT accuracy and TOBT stability. The first KPI is derived from comparing the TOBT with the ARDT. Again the mean square deviation between TOBT and ARDT is calculated for all flight of a day, where TOBT is taken at TOBT freeze time, i.e., 30 minutes before TOBT.

In the PoC testbed the estimate given by the ground handler will be a constant. The actual delivery time of the ramp services, however, will include some variation, e.g., dictated by the actual service requirements or by the service provider's resource trade-offs. Hence this parameter will be of interest.

The second parameter measures how stable the prediction mechanism of the ground handler is. For this purpose the average number of TOBT updates per flight is calculated.

Both parameters contribute directly to K1.

F. Ramp Service

Like the ground handler each ramp service provider wants to optimize his resource usage of both human resources as well as equipment.

For the evaluation of this business level objective two internal KPIs are devised, which do not contribute to K1 or to K2. They solely reflect the resource usage. Indicator 1 averages the usage of a type of resource over the period of a day, where usage is defined as percentage of resources occupied in comparison to resources available. One indicator is required for each type of resource.

The second indicator aims at avoiding overbooking. Per type of resource the number of occasions during a day are counted, when the service consumer tries to obtain resources beyond the available. Again, one indicator is required for each type of resource.

With regard to the overall A-CDM objectives two KPIs are required to evaluate the ramp service provider's performance: the arrival reliability and the service delivery duration. Both parameters contribute to K1.

VIII. FAILURE SCENARIOS

This section describes a number of cases that represent failures caused by malfunctions, performance shortcomings or security breaches that can affect the operation of A-CDM in an adverse manner. The SERSCIS tools and mechanisms are expected to handle these failure cases and improve the process performance of A-CDM even in the existence of failure conditions. The evaluation of these cases and thus the full validation of the SERSCIS results will be undertaken in project year 3.

The evaluation studies described in the deliverable at hand have a different purpose. Apart from proving the ability

to implement and reflect the process as described in chapter II), they should demonstrate that the testbed can actually perform failure cases and that meaningful KPIs have been chosen. The KPIs must enable the user to identify the effects of failures and to assess the impact of the SERSCIS mechanisms in handling the failure. Thus, the purpose of this is not to apply a huge number of failure scenarios but to focus on one or two cases that yield a representative assessment of the SERSCIS mechanisms and tools.

In order to cover the SERSCIS mechanisms as comprehensively as possible, mainly two distinctions of failure scenarios should be taken into account, the recurrence of threats and the phase when they occur along with the countermeasures provided by SERSCIS on the one hand and the type of security issue causing these on the other hand.

There are three basic types of threats or failures to be evaluated according to their recurrence and the countermeasure SERSCIS supports:

(M1) One-off threats or failures, for which SERSCIS can help to mitigate the effects.

(M2) Recurring failures, for which SERSCIS can support the mitigation by systematic adaptation.

(M3) System problems identified in modelling and prevented from happening by redesigning the system.

From a phenomenal cause point of view, failures can be induced by physical (C1) or by ICT related compromises (C2). The use case validation will cover both types. But in both cases the primary concern is the impact on and the usage of the ICT facilities to mitigate the threats.

A. Compromise of ramp service availability

In this scenario, the ramp service provider fails to respond to service requests in a timely manner or does not show up at all. In this case, the ground handler's request is not met with a reply containing the estimated completion time from the ramp service. After a timeout the ground handler could try to invoke the service a second time. If this does not succeed either, he would have to schedule and invoke the ramp service with an alternative provider. Once this provider replies to the service invocation with an estimated completion time, the workflow continues as described in Section V.A.

This event can be handled in two ways:

- As a one-off event that requires the selection of an alternative service provider
- From the point of view of a recurring event, which is counteracted by either blacklisting the specific service provider or by adapting the workflow such that it has more slack for late service delivery.

This scenario covers recurrence and countermeasures M1 and M2 and cause C1. It was used in the evaluation of both the run-time and off line SERSCIS components.

B. Passenger No-Show

A passenger who has checked in luggage does not show up for boarding. Consequently, his luggage needs to be unloaded. In its simple form, this is a scenario handled by countermeasure M1 (alternative workflow applied). It is caused by type C1.

This failure scheme could also be used for a massive distributed DoS attack if a huge number of passengers in coordination and on purpose do not show for boarding. Beyond the description above, that should also be detected by means of SERSCIS mechanisms.

This scenario was represented in all run-time tests (there is a passenger no-show in one flight in all scenarios used), leading to a small deviation from perfect KPI even in the 'sunny day' scenario.

In the off-line evaluation, the idea of an organised mass passenger no-show was also considered (creating a physical denial of service attack). This mass no-show could not be used in the run-time evaluation without a substantial extension of the PoC emulators for the Ground Handler and the Baggage Handler services. This is because the possible mitigation strategies involve changing the strategy for managing resources and computing the predicted TOBT (algorithmic adaptation), rather than at the agile SOA level.

C. ACISP Communication Delays

ACISP communication delays, caused by a denial of service (DoS) attack, can arise if the ACISP can be addressed from a sufficiently public network (e.g. the Internet), so an attacker can send too many requests (or possible a smaller number of malformed requests) in order to tie up the ACISP service's resources. It is caused by type C2.

To mitigate this threat, the ACISP can ensure their software stack is up to date, therefore reducing the opportunity for small numbers of malformed packets to cause a problem. They can also use a private network limited to the other airport stakeholders, although this may not be possible depending on how many stakeholders need access. Or they can deploy multiple redundant end points, and switch frequently so the attacker(s) will not know which endpoint to flood with malicious requests. These are all instances of M3.

This scenario with no mitigation was included in the evaluation of run-time SERSCIS components, to show how the KPI can be used to detect cyber-attacks as well as physical effects. The mitigation using redundant end-points could also have been implemented using the PoC testbed, but this was not done as it uses the same fail-over mechanism demonstrated in the compromised ramp service case. Other mitigation strategies could not be included in the PoC testbed as they would require explicit emulation of private/redundant communication networks, which was not yet implemented. In practice, these have to be included by design, so the vulnerability would need to be detected at the design stage via system modelling. However, the use of alternate networks (as well as endpoints) would need to be activated at run-time. So, even though prior modelling of the system is required (treating the threat as type M3), once this is done we then have to treat the threat as type M1/M2 in deciding when to activate the use of alternate networks or endpoints if a compromise is detected.

IX. DISCRETE EVENT SIMULATION EXPERIMENTS

The evaluation using discrete event simulation used the testbed to emulate a subset of the previously identified failure scenarios including types M1 and M2 above. The impact on identified KPI was measured (relative to a 'normal' or 'sunny day' scenario), thereby verifying that the testbed is able to emulate adverse behaviour, and that the impact can be characterised using the chosen KPI. Where the PoC testbed already provides an appropriate countermeasure based on the use of agile SOA, a further simulation was run to determine how this affects (improves) the emulated outcome and KPI.

This section lists the results obtained during the simulation runs. It describes the types of tests performed, shows the KPI values resulting from these runs and provides an interpretation of those. Several runs of the testbed were conducted for the evaluation. The runs represented different cases, one no-failure case and several degraded scenarios.

The simulation driver provides an interface (see Figure 6.) showing the progress of flights, and it is possible to inspect monitoring data for various application services (e.g., performance metrics) and SERSCIS components (e.g., SLA usage, etc.). Only if something goes wrong does the user receive any feedback through the DST interface (from WP5), after which the user must inspect the corresponding monitoring data to discover the cause, possibly aided by queries to a system-of-systems model. However, the emulation is designed to run in accelerated time (so each run does not take a whole day), and when this feature is used, there is very limited opportunity for user interactivity.

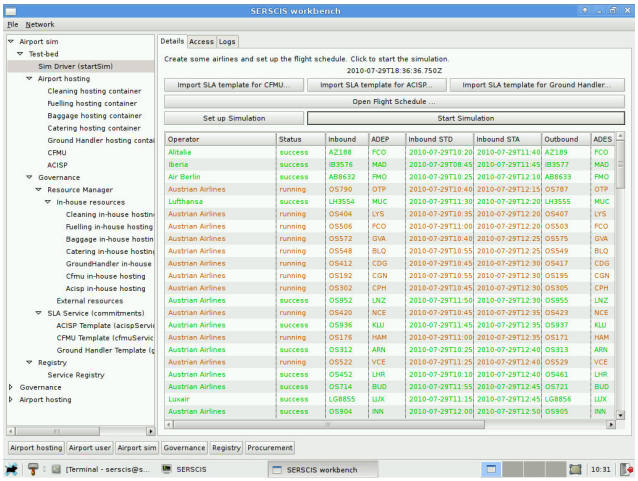

Figure 6. SERSCIS Graphical User Interface

The evaluation started with a 'sunny day' case, in which all service providers had a sufficient number of workers and hence always delivered. This was used to provide a starting point for further experimentation, and represents a best case scenario, although even the 'sunny day' case included a single passenger no-show to provide a 'background' signal in the measured KPI.

In the first approach to simulating a failure, the number of workers of one of the ramp services, specifically the baggage handler, was reduced step by step. Different runs were performed with ever-smaller number of workers until a threshold was reached when turn-around processes took a substantial time to complete. Once the threshold value was obtained, the policy of the resource manager was changed such that it could select an alternative service provider once the main provider failed. Specifically, once the baggage handler failed to respond to a service perform request due to a lack of workers it was considered failed. In this case it was replaced by another baggage handler with a sufficient number of workers.

The second approach to simulating a failure affected the communications of the ACISP. Similarly to above, the delay in communications to and from the ACISP was increased step-by-step until degradation in the simulation KPIs was observed. This experiment was used to model a denial of service attack on the airport network.

A. KPIs used in the evaluation

In section VII.F, two KPIs to evaluate the performance on a ramp service provider level are listed, his arrival reliability and his service delivery duration. In the proof-of-concept evaluation the second KPI is a constant and disregarded. The first KPI on the other hand is taken into account to show the effect of a reduced number of workers. It is assumed that the reliability decreases if the number of workers available at a service provider is reduced. In the testbed this is measured by the number of "perform attempts" issued to the service provider. If a service provider has sufficient resources, every flight requires exactly one perform attempt that is honoured by the service provider; i.e., the number of perform attempts must be equal to the number of flights. If the provider cannot immediately honour a perform attempt due to a lack of workers, the perform attempts will be repeated. Thus the number increases beyond the number of flights.

The ramp service performance also has an effect on the ground handler's KPIs. As described in section VII.E, two KPIs characterize this performance, TOBT accuracy and TOBT stability. Since the actual delivery time of the ramp services is a distribution with a certain variation, e.g., dictated by the actual service requirements or by the service provider's resource trade-offs, TOBT accuracy will decrease with a reduction in the number of workers at the ramp service provider. TOBT stability expresses the number of updates to the TOBT required for each flight. In a sunny day scenario this number should be 1 or close to it, i.e., once a TOBT is issued it will not be changed. In degraded scenarios, however, a ramp service will deliver late due to a lack of workers. In the testbed the ground handler re-issues a TOBT whenever the estimate deviates from the previous value by more than 10 minutes. Hence the longer the ramp service delays its service delivery the more TOBT values need to be issued for a flight.

Both above-mentioned KPIs affect the number of take-offs outside the slot-tolerance windows (STW), an overall CDM KPI (K1 as listed in Section VII.A). The value will increase in a ripple on effect of the ramp service provider's inaccuracy. If the ramp service provider fails to show up on

the initial perform request, there is a risk that he delays the turn-around of a flight and causes it to miss its slot. This effect, however, might be countered in part by an available slack in the turn-around process.

A policy change that allows to replace the service provider with an alternate in case he fails to respond to a perform request must reverse the above effect. Choosing an alternative service provider when the primary provider failed, will replace the overall service delivery reliability and thus result in less take-offs outside the STW.

Another KPI is applied to evaluate the overall CDM system's performance, the average number of slots issued per flight. Obviously, in the ideal case one slot is issued for a flight and this one is used subsequently. In less ideal situations delays in the turn-around prevent a flight from meeting its slot. Thus a new slot has to be issued, potentially wasting the previous one if it cannot be claimed by another flight. The longer the delay of a turn-around, e.g., induced by a lack of workers at a ramp service providers, the more slots must be issued for a flight[3].

B. Applied scenarios

In the beginning of this section, a general description of the scenarios was given. This section provides more details on the various scenarios and the changes for the different failure cases.

All scenarios use a schedule of 124 flights to be turned around during a day. All of those are regulated flights, i.e., all require a slot.

Apart from the non-failure case, three degraded mode cases are listed and assessed below.

The first case, the 'sunny day', provides sufficient resources for all ramp service providers. Hence none of the flights experiences a delay in turn-around.

Case 2 is characterized by a reduction in the number of workers of the baggage handler to 23. In this case the baggage handler fails to honour several perform attempts and the flights experience substantial delays.

In case 3 the number of workers is further reduced to 18. Hence even fewer perform attempts are honoured.

In case 4, a second baggage handling resource was introduced (i.e., the Ground Handler starts with two SLAs for the provision of baggage handling services with different suppliers). Now if the primary baggage handler fails, an alternative service provider can replace it. If this arises as a one-off problem it can be handled by the service orchestrator component (mitigation type M1 as defined in Section IV), though some delay will still be experienced. If the primary supplier persistently fails, it is better to manage the situation by excluding it from further use (mitigation type M2). This was done by attaching the policy shown below to the individual baggage handler resources (as seen by the Ground Handler). This policy sets the condition of the service to 'failed' if there is more than one failure, and deregisters the service so preventing it being offered to the orchestrator:

This addresses the immediate problem of a failing supplier, but it reduces the number of available options for baggage handling to one. A further policy is therefore needed, attached to the resource manager, causing the resource manager to procure a new SLA with a replacement baggage service provider (referred to by the SLA template provided).

Finally, case 5 implements a simulation of communication delays to demonstrate the effects of a denial of service attack on the ACISP. Due to the slow rate of communications, a number of flights take off outside their slot windows. No mitigation for this was considered in the run-time tests, as the only one that could be handled by the PoC emulators was to have redundant ACISP endpoints, which duplicates the mechanisms tested in Cases 1-4.

C. KPI results

TABLE IV. KPI RESULTS

KPI	Case 1	Case 2	Case 3	Case 4	Case 5
Baggage perform attempts	249	556	961	266	249
Average TOBT error	4 min	14 min	49 min	4 min	4 min
Average TOBT updates per flight	1	1,5	2,7	1	1
Average number of slots issued	1	1,5	2,7	1	1
Take-offs outside STW	0%	15%	31%	0%	13%

The results shown in Table IV clearly indicate that the chosen KPIs are meaningful for the testbed and the scenario and that verification and validation of the testbed succeeded.

The KPI "Perform attempts" was expected to increase if a service provider does not have sufficient resources to honour all requests in parallel. In this case some of the requests must be repeated, which means a larger figure. When introducing the possibility to choose an alternative provider in case the first one fails to honour requests, the total number of perform attempts should decrease again.

This is exactly the behaviour of the testbed. Case 2 and also case 3 exhibit a significantly larger number of perform attempts than the sunny day case 1. With the introduction of an alternative service provider in case 4, the number of request drops close to the value of the sunny day case again. Note that it is still slightly larger than in the sunny day case, because additional perform requests are issued (and not honoured) while the alternative provider is being set up. Hence the KPI provides meaningful characteristics of the testbed and the testbed shows the expected behaviour.

The TOBT-related KPIs reflect the quality of service delivery by a ramp service provider. With a decreasing

[3] In the testbed the ground handler uses a simple strategy to update the TOBT. A new estimate for TOBT is calculated, and the TOBT is updated if the new estimate is more than 10 minutes after the previous TOBT. Note that in the current simulation every TOBT change automatically results in the issuance of a new slot. For this reason, currently the number of slots per flight is equal to the number of TOBT updates. The implementation of this behaviour will be re-assessed for the final validation.

number of workers in cases 2 and 3, the TOBT accuracy decreases as well and the required number of updates to this value per flight increases accordingly. When the ground handler has the option to choose an alternative service provider in case 4, the trend reverses and case 4 delivers the same performance as the sunny day case 1.

Similarly, the number of slots issued per flight increases with the number of TOBT updates per flight. In case 4, in which TOBT does not get updated, only one slot is required as in case 1.

The last KPI, which was assesses in the testbed evaluation, is the percentage of take-offs outside the slot-tolerance window (STW). In the case of a sufficient number of workers at all service providers (case 1), none of the flights should miss its slot[4]. Hence the KPI must be 0%. With turn-arounds being delayed due to an insufficient number of workers at one of the service providers, flights will miss their slots and take off outside the STW. For this reason the value increases to 19% in case 2. When an alternative service provider steps in to take over the tasks from a failed provider as in case 3, turn-arounds are on time again. The percentage of missed slots falls back to 0% again.

In the event of communication delays with the ACISP we see the percentage of takeoffs outside the slot tolerance window increase in proportion to the delay. This is caused by delays in communications resulting in windows of opportunity to be missed.

The KPI reflects these behaviours as expected and thus also indicates a correct behaviour of the testbed.

X. CONCLUSIONS

This paper describes a base metric model that provides a uniform abstraction for describing service behaviour 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 analysed the application service that it is offering, defines a metric ontology to describe measurements of the relevant service behaviour. This ontology should refer to the SERSCIS base ontology, and provide subclasses of the base metrics to describe each relevant aspect of service behaviour. 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.

The evaluation of SERSCIS using discrete event-based simulation and KPI-based definition of behavior also proved to be a good indicator of the approach. KPIs were used as a starting point for run-time monitoring and mitigation strategies. Using an appropriate set of KPIs the behaviour of the system can be monitored effectively and efficiently. Failures of individual services can be detected and – given that mitigation strategies are implemented – their effectiveness can be observed as well.

[4] The limited capacity of taxiways and runways might cause flights to miss their slots despite a timely turn-around, but this is not modelled in the testbed.

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..

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

[9] WSDM-MOWS Specification. www: http://docs.oasis-open.org/wsdm/wsdm-mows-1.1-spec-os-01.htm (Last accessed Feb 2012).

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

[11] M. Surridge, A. Chakravarthy, M .Bashevoy and M. Hall-May. "Serscis-Ont: A Formal Metrics Model for Adaptive Service Oriented Frameworks". In: Second International Conference on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE), 2010.