A Survey on Fault-Models for QoS Studies of Service-Oriented Systems

Philipp Reinecke[1], Katinka Wolter[1], and Miroslaw Malek[2]

[1] Freie Universität Berlin
Institut für Informatik
Takustraße 9,
14195 Berlin
{philipp.reinecke, katinka.wolter}@fu-berlin.de
[2] Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin
malek@informatik.hu-berlin.de

Compilation date: Thursday 25th February, 2010 – 08:57

Abstract. This survey paper presents an overview of the fault-models available to the researcher who wants to parameterise system-models in order to study Quality-of-Service (QoS) properties of systems with service-oriented architecture. The concept of a system-model subsumes the whole spectrum between abstract mathematical models and testbeds based on actual implementations. Fault-models, on the other hand, are parameters to system-models. They introduce faults and disturbances into the system-model, thereby allowing the study of QoS under realistic conditions. In addition to a survey of existing fault-models, the paper also provides a discussion of available fault-classification schemes.

1 Introduction

In order to study and address Quality of Service (QoS) issues in service-oriented systems, we need a model of the system in question. Such a *system-model* allows us to study important properties of the system. Systems can be modelled at various levels of abstraction, ranging from abstract mathematical frameworks such as stochastic processes or queueing networks to system testbeds, i.e. physical systems equipped with measurement and experimentation infrastructure. Ideally, models at different abstraction levels should be used, as different models can often complement each other. A queueing-network model of a system, for instance, may be used to efficiently study a large space of parameters, and thereby arrive at general conclusions. A testbed-model of the same system, in contrast, enables measurements under realistic conditions, which serve to validate the more abstract model, and to improve the quality of the conclusions by providing realistic model parameters.

Irrespective of their abstraction level, all system-models allow us to study properties of the system. In the following we therefore subsume the whole range

of models between abstract mathematical models and system testbeds under the general concept of the system-model of the service-oriented system under study, and refer to the actual type of model only where necessary or illustrative.

When studying QoS issues we are particularly interested in the behaviour of the system under various common faults or disturbances. With our system-model we can assess this behaviour using *measures* defined on the model. For instance, in a queueing-network model we may compute job completion times, while in a testbed we may measure response-times.

In order to use a system-model to study the effect of faults on a system, we must be able to introduce models for these faults into the system-model. Since with some model classes the system-model may change significantly when introducing a model for a fault we require that we can obtain the same measures with the same interpretation from the system-model, regardless of the employed fault-model.[3] That is, using the terminology of functional and non-(or extra-) functional behaviour, we require that the system-model maintains the same functional behaviour and the same system structure when we change the fault-model.

This clear distinction between the fault-model and the system-model is common in fault-injection experiments for dependability benchmarking, where one explicitly describes a fault-load that the system is subjected to [2, 3]. We apply this concept to system-models in general, by describing the factors that affect QoS by fault-models, which we treat as parameters to our system-model. Such parameterisation may be as simple as setting a service rate in a queueing-network model, or a constant delay in a testbed. In this case, the parameter to the system-model is the constant service rate or the constant delay. However, service rates or delays may vary over time. If we have a model for these variations, we can replace the constants by this model. In doing so, we set the fault-model parameter of the system-model to be the model expressing service-rate or delay variations.

This survey paper offers two major contributions to the study of QoS issues in service-oriented systems. First, we provide an overview of the fault-models available in the literature, using a fault-classification scheme based on architectural properties of a SOA system. Second, we review schemes that have been proposed for classifying faults in SOA systems. The first part (Section 2) aims to give the researcher interested in QoS issues a structured overview of available fault-models that may be used for parameterisation. It may also serve to direct measurement efforts to those components of SOA systems whose behaviour is as of yet poorly reflected in fault-models. Our review of fault-classification schemes in the second part of the paper (Section 3) is intended to give a guideline for a

[3] E.g. introducing response-times modelled by phase-type (PH) distributions (cf. e.g. [1]) into a CTMC model will lead to a drastically different model, as this is accomplished by replacing a single transition by the Markov chain underlying the PH distribution. Still, the same measures, such as first-passage time, can be computed for the original and the modified model, and have the same interpretation for both.

structured approach to building and parameterising system-models that are not limited to one specific system, but instead reflect general properties.

In the remainder of this section we will first give a definition of the concept of a fault-model, as we employ it in the following sections. We then describe our approach to structuring the survey.

1.1 Fault Model Definition: What Constitutes a Fault Model?

We regard fault-models as parameters to a system model that influence the modelled system's QoS. In order to identify suitable fault-models, we need to clarify our concept of a fault-model. The literature provides an abundance of variations on the term, however, for the most part these are used to describe certain application-specific aspects (e.g. 'stuck-at-X' for hardware faults, or specific programmer errors for software faults), rather than providing a definition of what a fault-model actually *is*.

Going back to the basic concepts of fault-tolerant computing, we can derive just such a definition. Following [4], we employ the notion of systems (servers) that implement a set of functionality (the service) to be used by other systems (users, clients), which in turn may be servers (and thus provide a service) to further systems. The distinction between server, service and client is generic enough to be made in system-models in general, although this may not always be as obvious as with e.g. a queueing-network model, where jobs are the clients, server stations are the servers, and the service is provided by transition through the server stations. The service provided by a server is *correct* if it is in accordance with the specification for the service, which may define both functional and non-functional properties. Based on [5], we call deviation from the service specification a *service failure*, and the cause of a failure a *fault*.

Now, as the server provides its service to a client, which again can be thought of as a server to other clients, a service-failure in the server may be the cause of a failure of the service the client in turn provides to its own clients, and hence the server's failure is a fault for the client. Different failures of the server (i.e. different deviations from correct service) constitute different faults to the client and may result in different client failures. Our notion of fault-models as parameters to a system-model follows from this observation: By changing the characteristics of the deviations from correct service we can induce different faults in the client system. Consequently, to model the faults that may affect a system we need a description of the failures of the server(s) that the system under study depends on. *A fault-model, in a general sense, should therefore consist of descriptions of the server, the service, and the characteristics of the deviations from correct service that constitute a failure.* Note that the latter often depend on the service as well. With response-times, for instance, there may be a threshold beyond which service-responses become useless to the client; this threshold, however, may vary between clients. We thus relax the requirement that the model describe the characteristics of deviations such that we also allow models which do not explicitly provide this description, but enable the derivation of characteristics that amount to a failure.

We divide the large set of fault-models that fit the above definition into the following three types, based on how the models describe the characteristics of the failures (in the server system) or, equivalently, faults (in the client system):

Bernoulli Trials. With these, the fault is described as a Bernoulli-distributed random variable. That is, the occurrence of the fault is defined by a probability p. Typical examples for this type are fault-models that reflect service availability/unavailability, e.g. a service is available with a probability of $p = 0.95$.

General Random Variables. Here, fault occurrence is described by a random variable with a distribution that is more general than Bernoulli. Ideally, the distribution should be described by a distribution function (cumulative distribution function, CDF), a complementary CDF (CCDF) or a probability density function (PDF), but we will also include less complete characterisations, e.g. by a mean or a quantile. Examples for this kind of fault-model are response-time distributions. As different operating conditions may affect the distribution, more elaborate models of this category allow parameterisation. For instance, response-times may vary with system load, and this may be expressed by individual response-time distributions for the different load-levels.

Stochastic Processes. These provide a characterisation of faults over time, capturing e.g. correlation. We use a very loose definition of a stochastic process, in that we only require that the fault-model contains a description of states and of state transitions. The states may provide arbitrarily complex characterisations of a fault, while the state-transitions serve to define the time-dependent behaviour. For instance, the classical Gilbert-model (cf. e.g. [6]), which describes a no-loss and a loss-state and transition probabilities between both, is a stochastic process. Note, however, that we do not put any restrictions on the distributions characterising state-transitions. In particular, we do not limit the focus to Markovian models.

Let us illustrate our definitions using a simple example: In [7] we presented Acyclic Phase-type (ACPH) models of the transmission delays experienced when sending SOAP messages over a network link with IP packet loss. Messages were transmitted over WSRM (Web Services Reliable Messaging, [8]), which provides a reliable message transport within the SOAP stack. Furthermore, the SOAP transport used by the WSRM layer was HTTP over TCP, the latter of which provides fault tolerance for IP packet loss.

In this case, the server is a link capable of reliably transmitting SOAP messages, the service is end-to-end message transmission, and the client is an application that depends on timely SOAP message transmission. We want to use this fault model in a queueing network model of a SOA system. The definitions of the server, service and client are required to determine where we can use this fault model in the SOA system-model. E.g., we can obviously not use it for a Web Service, but we can use it for the network link required to invoke the Web Service. The behaviour of the service of the link is described by ACPHs, which are general random variables according to our above distinction. Based on these,

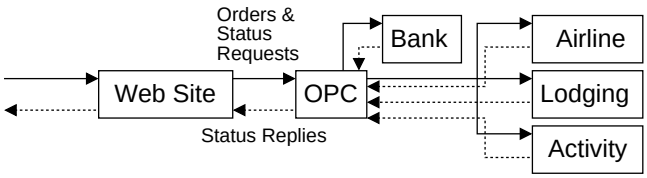

Fig. 1. Functional structure of the SUN Java Adventure Builder reference implementation.

we can define when a fault occurs by e.g. defining a threshold consistent with the requirements of the application. However, it may be preferable to use the distributions directly, thus leaving the exact definition of a fault implicit, and only observing the effects of faults on the application.

1.2 Structure of the Survey

Before embarking on our survey of available fault-models, we need to define a structured approach to the review of the literature. We want to obtain a survey that is as exhaustive as possible, in that it does not exclude relevant fault-models. Fault-classification schemes help in this regard, since they give guidelines as to which faults we may observe in a SOA system, and hence have to find models for. We review five fault-classification schemes for service-oriented systems in Section 3. Unfortunately, as will be illustrated there, these schemes do not provide clear-cut criteria for many common faults, which shows in overlaps between fault categories. Furthermore, the schemes tend to consider only abstract fault classes, and thus the guidelines they provide are incomplete.

For these reasons, we employ a recursive system-based approach: We start with the typical structure of a system and try to find fault-models for the behaviour of each of the components of the system at a high abstraction-level. We then look at the typical structure of each of the components and repeat this recursion until we have obtained suitable fault-models or until further recursion becomes unlikely to yield further insight.

To this end, we start with the architecture of the SOA reference application SUN Java Adventure Builder [9], which we consider a typical SOA system (Figure 1): The system implements an online travel agency that users can interact with using a web site, accessible by HTTP over the Internet. The web site back-end calls four Web Services, using the SOAP protocol, which again translates to HTTP transactions over the Internet. Each of the services shown in Figure 1, as well as the web-site back-end, corresponds to a physical system implementing the functionality of the provided service. In the case of the web-site, this system typically consists of an HTTP server running on top of an operating system, which in turn depends on the underlying hardware. The Web Services are deployed within application servers, which again require an operating system (and hardware). Frequently, services rely on storage systems such as databases for their operation. Furthermore, the hardware beneath the operating system need

not be a physical machine, but may instead be a virtual machine that shares the physical hardware with several other virtual machines. Faults in each of these (and their sub-systems) may affect quality of service experienced by the user. A closer look at the communication links between the components shows that a similar dissection can be performed here. Faults can occur in the Web Services stack and in the Internet underlying the WS stack. In both cases, the infrastructure and the transport itself are of interest.

We choose this recursive approach to structure our survey because it considerably extends the number of potential sources for fault-models, by avoiding the limitation to studies of SOA systems. For instance, Internet behaviour has been studied independently, and fault-models derived there are useful in SOA systems, which rely on message transmissions. On the other hand, the structure implied by the recursion helps to limit the amount of studies to be considered.

2 Available Fault-Models and Data

We will now start the first part of our survey, an overview of fault-models available in the literature. We start with individual Web Services and typical subsystems they rely on. We then consider the communication part, using the same method. Our recursive approach results in the following structure:

1. Web Service
   (a) Platform
      i. Application Server/SOAP Framework
      ii. Virtualisation
   (b) Storage/Databases
2. Communication
   (a) Web Services Stack
      i. Infrastructure
      ii. Message Transport
   (b) Internet
      i. Internet Infrastructure
      ii. Transport

In terms of the definition of a fault-model, each item in this list describes a server that may deviate from its intended service. Thus in each of the following subsections we consider studies that describe, or at least allow us to derive, characterisations of the deviations from correct service for the respective server. In doing so, we first try to identify Bernoulli models, then continue with general random variables, and finally stochastic-process models. A summary is given in Tab. 1 on page 16.

2.1 Web Service

At the highest level, we study a system consisting of a Web Service and a user (human or machine), e.g. in Figure 1 the OPC is a Web Service that is accessed

by the Web Site. In our model of this system, the Web Service is the server whose failures constitute the faults that affect the client. Thus we require fault-models that reflect the behaviour of Web Services, as seen by their users. With our focus on quality-of-service aspects, we are interested in faults that manifest as either unavailability of the service (i.e. the service does not respond to or rejects requests) or insufficient speed of responses.

Bernoulli models for the availability of Web Services can be derived from the study of the availability of Web Services in a publicly-accessible UDDI repository in [10]. However, the paper only states the probability of a Web Service in the UDDI being available (0.84).*

[FIXME: There should be more, and more recent, studies on this.]

General random variables for the response-times of Web Services may be defined over a set of services, or over invocations to one specific service. The observation in [10] that the 96% quantile is at 2 s for the set of Web Services analysed in the study is an example for the former, while the finding in the same paper that response-times for the Amazon and Google Web Services vary between 502–510 ms and 329–1046 ms, respectively, is an instance of the latter. Another description of response-times as a random variable over invocations to the same service may be derived from [11], where histograms for two public Web Services are provided. We note that neither of these studies gives explicit fault-models, and furthermore, that there is no description of the shapes of the probability distributions in [10].

More elaborate models still describe response-times by random variables, but allow additional parameters to reflect operating conditions. Fault-models of this type may be derived from the studies in [12, 13, 14, 15]: In [12] and [13] E-Commerce applications are studied in testbed environments. [12] contains 90% quantiles for response-times at load levels of 50, 100, 150, and 200 clients simultaneously accessing the application. The results are split by the services that make up the application, and by two different usage profiles. Depending on the load level and the usage profile (Browsing/Ordering), the Flights, Credit Check and Process Order services exhibit 90% quantiles of response-times of 150–200 ms/150–800 ms, 120–550 ms/150–700 ms, and 150–800 ms/200–900 ms, respectively. A characterisation of load-dependent response-times by averages is given in [13]. For load varying between 10 and 100 users, mean response-times increase from 500 ms to 1800 ms (flattening out at 60 users due to saturation). The study of Web Services mean response-times in [14] shows that, starting at 140 requests per second, response-times rise steeply to slightly above 7000 ms at 160 requests/s, where a plateau is reached (due to saturation). Unfortunately, due to the scaling of the graph, response-times for less than 140 requests per second are not discernible. Finally, in [15] we provide Acyclic Phase-type (ACPH) models for the response-times of the SUN Java Adventure Builder at load levels of 10 and 50 users. Furthermore, the effect of IP packet loss on response-times is studied. It is shown that load and packet loss both increase mean response-times and the variance of the response-times distribution.

We note that, with the exception of [15], the studies give very sparse descriptions of response-time distributions. While one might assume that response-times

follow an exponential distribution (in which case the mean would be sufficient for parameterisation), this seems doubtful from the findings in [15]. To the best of our knowledge, the ACPH models in [15] are the only explicit fault-models for Web Service response-times, and even these only describe response-times by random variables and cannot capture variations over time. We thus observe that so far the behaviour of Web Services has not been captured sufficiently well by fault-models. Therefore we now perform the next step in the recursion, by splitting up the Web Service into the platform on which it is deployed and the storage it requires for operation.

2.1.1 Platform We use the term 'Platform' to refer to the entire physical system implementing the Web Service. That is, we now no longer view the Web Service as an abstract entity that responds to requests, but instead consider the actual computer system(s) involved. While we are not aware of studies that deal specifically with the physical systems for Web Services, one may argue that these systems are similar to other complex computer systems. Based on this premise, we extend our view to include fault-models for computer systems in general. However, we should keep in mind that these models are derived from the behaviour of physical systems that might differ from those underlying Web Services.

Bernoulli and general random-variable models of the end-user availability of large systems on the Internet can be parameterised based on [16]. Analysing service-availability for three web sites, the authors report values ranging from 0.9305 to 0.9994, depending on the assumed locations of faults. In particular, they state that local problems have a strong impact on availability. The presented graph of response-time over availability may serve to parameterise a random variable representing timing-failures that amount to service failures.

Simple stochastic-process models for faults can be thought of as consisting of an 'ok' state and a 'failed' state. In the 'ok' state, the modelled system provides its intended service, while in the 'failed' state the service is not available. Parameters for this kind of model can be derived from [17], where failure data of systems in the Los Alamos National Laboratory (LANL) from the period 1996–2005 is analysed, from [18], where three commercial Internet services are studied, and from [19], where the reliability of Internet hosts is analysed.

In the model, sojourn times in the 'ok' and 'failed' states are characterised by the time-to-failure (TTF) and time-to-repair distributions (TTR), respectively. For the TTF analysis, the authors of [17] pick one system out of their data set. They treat early system life and remaining life independently, since this system has significantly higher failure-rates during early system life. Furthermore, as the chosen system in [17] is comprised of 49 physical nodes, they consider failures occuring in one single node and in the whole system. During early life, the TTF distribution for the single node has a squared coefficient of variation of $c^2 = 3.9$ and can be described by a lognormal distribution, while no standard distribution provides a good fit for data from the system-wide view. The latter finding is due to the high number of simultaneous failures, which imply failure

interarrival-times of zero. During remaining system lifetime the TTF distribution for the single node has $c^2 = 1.9$ and is approximated well by Weibull (shape parameter 0.7) and Gamma distributions. Weibull (shape parameter 0.78) and Gamma distributions also describe TTF distributions in the system-wide view. The TTR distribution is obtained over all systems in the data set. This distribution has mean 355 min, median 54 min and $c^2 = 187$ and is approximated by a lognormal distribution. Mean and median repair times depend on hardware type. Furthermore, repair-time distributions also depend on the root-cause of the failure. Combined with the root-cause analysis also provided in the study, this may be used to further parameterise repair-time distributions. We note that, with the exception of the shape parameters for the Weibull distributions, the paper does not provide distribution parameters explicitly, but shows the CDFs of the data and the fitted distributions. Time-to-failure and time-to-repair distributions are often characterised simply by their means, i.e. the Mean Time To Failure (MTTF) and Mean Time To Repair (MTTR). Assuming exponential TTF and TTR distributions, MTTF and MTTR are sufficient to specify a stochastic-process modell. MTTR values for Internet services can be found in [18]. The study in [19] provides distributions of MTTF and MTTR values over Internet hosts. These distributions are described both by statistical properties and plotted densities and distribution functions. Neither distribution is well-described by an exponential distribution.

We note that a stochastic-process fault-model for a Web Services platform may be derived from [17], if one deems the systems studied therein sufficiently similar to the systems that Web Services are typically deployed on. The study in [18], whose data sets come from systems supposedly more similar to typical Web Services, provides only limited insight, in that the TTF distribution is not studied at all, and the TTR distribution is only characterised by mean values, which seems insufficient, considering the finding in [17] that neither the TTF nor the TTR distributions are exponential. The analysis in [19] provides some insight into the distribution of MTTF and MTTR values over hosts, but does not characterise TTF and TTR distributions for the hosts in more detail.

We now apply the next step of recursion, by dividing the platform of a Web Service into its components. We consider the application server and SOAP framework, and virtualisation software. One further component of interest would be the operating system.

2.1.1.1 Application Server/SOAP Framework The software implementing Web Services is typically deployed within an application server and a SOAP framework, which provide the run-time environment required by the service to run and communicate with other Web Services. We thus now consider the application server and SOAP framework as a server, and the Web Service as a client to this server. With respect to fault-models the service is then the run-time and communication functionalities required by the service. Again, both unavailability (which would in turn render the Web Service unavailable to its clients) and timeliness are relevant properties.

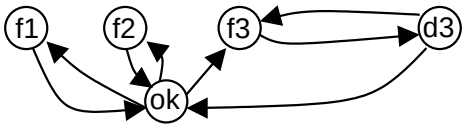

Fig. 2. CTMC fault-model (incomplete) for virtual machine reboots.

A stochastic-process model for application-server failures may be derived from the Markov reward model for the availability of a dependable application server presented in [20]. To this end, we consider the application server unavailable (unable to provide the run-time environment for the Web Service) if the high-level model is in one of the two 'failure' states. Transitions into and out of the 'failure' states are then described by exponential distributions, whose rates have been computed based on sub-models which are parameterised partly based on measurements.

Models of SOAP framework throughput (served requests per second) as functions of time under heavy load might be obtained from the study of the SOAP framework Axis 1.3 in [21]. There, it is shown that memory leaks in the framework result in decreasing throughput, and even application-server crashes. From the figures, it appears that, depending on the load, linear functions with a strong negative slope or exponential functions may approximate the throughput degradation well. Unfortunately, the authors have not attempted fitting analytic functions to their data. We note that, although it appears likely that the causes of these memory leaks have been addressed in later versions of Axis, one may still employ these observations as guidelines when studying the effects of memory leaks, and software aging in general.* Note that the data given in [21] represents only single observations at each time-point. Nonetheless, it may indicate how a stochastic process could describe the effects of memory leaks.

[FIXME: Is the model a stochastic process or a random variable or something else?]

2.1.1.2 Virtualisation In recent years, virtualisation has become an important technology, as it helps to increase utilisation of expensive hardware and to reduce power consumption. When virtualisation is used, many server instances may run on a single physical machine. With service-oriented systems, this implies that many services might be running on the same physical system, each within its own application-server environment. The required virtualisation software, however, adds another layer of complexity, and hence another source of faults, with respect to both timeliness and availability. While we are not aware of studies focussing on virtualisation software behaviour, it has been pointed out in [22] that faults, and fault-prevention by rejuvenation of the virtualisation software may require that virtual machines be stopped and restarted. The way in which these restarts affect the virtual machines depends on the virtualisation software, and may vary from normal reboots on physical hardware.

A stochastic-process model for the effects of virtual-machine reboots can be derived from [22]. The model (Figure 2) has one 'ok' state, in which the virtualisation software has no effect on the system running in the virtual machine, and

three failure states ('f1' through 'f3'), in which the virtual machines are down. Additionally, there is one degraded state ('d3'), in which the machines are up, but provide service at reduced throughput. The three failure states reflect the different machine resume strategies considered in [22], which differ in their effects on the systems running in the virtual machines. Failure states 'f1' and 'f2' represent 'resume from disk' and 'resume from RAM', and these strategies differ only in the incurred downtimes. The 'f3' state is used for the 'cold reboot' strategy. This mechanism, besides incurring a delay, also results in a state where throughput of the systems inside the virtual machines is degraded, since caches are emptied during machine reboot. The authors of [22] provide measurements for the downtimes, and also for the amount of throughput degradation. Assuming that downtimes follow an exponential distribution, we may use these times as parameters in our model. Throughput degradation provides a guideline to describe behaviour in the degraded state. Note that the model is incomplete, in that for some transitions (notably the ones *into* the failed states) no characteristics are given. This is due to the fact that [22] does not focus on the failure behaviour of virtualisation software and instead concerns the effect of the resume strategies.*

[**FIXME:** Check the TDSC version of the paper for the migration stuff.]

2.1.2 Storage/Databases Most services rely on extensive data manipulation and storage, e.g. for storing and retrieving service or business-process state. From the perspective of a service-oriented system, the service of databases, or storage in general, is to store and retrieve data.

Bernoulli models for the service availability of database systems can be derived using [23] and [24]. The authors of [24] give availability estimates for a database system in normal operation and under the influence of operator faults. Depending on hardware and software platform, database version and database configuration options, availability observed at the server/client, respectively, varies between 74.2/68.7% and 93.5/83.9%, with database configuration having the largest impact. The probability of low-level hardware/software faults causing database failures, studied in [23], may be used to parameterise a model for database faults, if the probability of low-level faults is known.

A random-variable model for the throughput of database systems without and with injected operator faults is the second type of model that can be derived from [24]. Unfortunately, throughput is only specified by mean values. Throughput depends on hardware and database configuration, varying between 1411 and 4394 transactions per minute without faults, and 896–3043 transactions per minute with faults.

Recovery-times of a database system with two types of automatic recovery are provided in [3]. The data is given as mean values for the recovery-time under operator faults and with different database configurations. These may be used as a parameter in a stochastic-process model, however, there is no characterisation of the distribution of the recovery-times, and no times-to-failure are given.

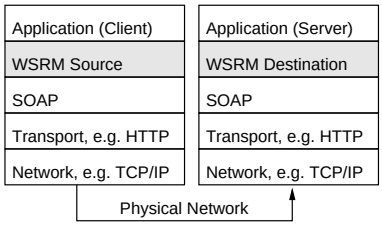

Fig. 3. SOAP Stack

The papers only present a first glimpse into the behaviour of storage systems. While these data may be used to derive fault-models, this area could greatly benefit from further analyses of such systems.

2.2 Communication

As service-oriented systems rely on communication between services, faults in the communication adversely affect service quality. We split this part into faults in the Web Services stack and in the Internet, where the former subsumes all technologies specific to Web Services, and the latter has a broader focus on the faults in the communication technologies that enable Web Services communication, but are not specific to them.

2.2.1 Web Services Stack (Infrastructure) We start with the behaviour of the Web Services Infrastructure, i.e. those components of a service-oriented system that enable service communication, such as UDDI Repositories, which allow publishing and discovery of services.*

[FIXME: PR: Are there any papers on this? UDDI part of [10], perhaps?]

2.2.2 Web Services Stack (Transport) Next, our focus lies on the Web Services stack, i.e. on the communication protocols specific to Web Services. In Figure 3 we show an instance of the SOAP stack; we now consider all layers above the lowest two (HTTP and TCP/IP).

The Web Services Reliable Messaging Protocol (WSRM, [8]) is an addition to the SOAP stack that provides reliable SOAP message transmissions over unreliable SOAP transports. A set of ACPH models for the message-transmission times for several WSRM timeout strategies and several levels of IP packet loss are described in [7]. The measurement data shows that, even though both TCP and WSRM mask the effects of IP packet loss, there are still characteristic patterns of the transmission-time distribution (viz. spikes at the values of the TCP RTO timeout) in the message transmission times, which are also captured in the models.

[FIXME: Note something on SOAP overheads?]

*

We note that, so far, only very few models for the behaviour of communication components of service-oriented systems are available. Since Web Services

typically communicate over the Internet, or networks using Internet technology, their communication is affected by the behaviour of these networks. This has long been a field of intense study, and thus we may hope to find appropriate models that could be used in the study of SOA systems. As with the Web Services stack, we divide this part into infrastructure, which refers to those components enabling the communication, and transport, which is the actual transmission of upper-layer messages or data packets.

2.2.3 Internet (Infrastructure) Here, we consider two important infrastructure elements of the Internet, viz. the Domain Name System (DNS) and the routing infrastructure. As both communication with the Web Services infrastructure and between services relies on DNS lookups of the systems the services are deployed on, faults in the Domain Name System may affect both availability and timeliness of Web Services communication: If name lookup fails, then a Web Service may be invisible (and thus unavailable), and delays in lookups lead to delays in the communication.

Bernoulli models for DNS server availability can be parameterised using [25], where detailed availability values are given for name servers. In particular, the mean availability over all analysed servers is 0.9785, with median 1, i.e. for most servers no unavailability period could be observed, while those with outages tend to have rather low availability.

Random-variable descriptions of DNS lookup response-times from correctly configured servers, and from servers with one of three different misconfigurations can be derived from [26] The paper provides CDFs of these random variables. In particular, non-responding servers increase mean response-times from about 60 ms to 3 s, and up to 30 s in extreme cases. The paper also analyses the frequency of misconfigurations at various top-level domains, and thus gives the probability of hitting a misconfigured server, if domains are chosen at random. Unfortunately, no information on temporal dependencies, or variations in responses for individual addresses is provided.

A stochastic-process model for service availability of DNS servers is the second contribution in [25]. The model consists of an 'ok' and a 'failure' state, with the former handling DNS requests as expected and the latter not responding to requests. The transitions between both states are described by the time-to-failure and time-to-repair distributions, which the paper characterises by mean, standard deviation and median values, and by graphical CDFs. The parameters differ depending on the analysis method; however, it appears that time-to-failure distributions are hypo-exponential, while the recovery-time-distributions are hyper-exponential. For instance, one TTF distribution has mean 125.9 h and standard deviation 99.1 h, while the associated TTR distribution's mean and standard deviation are 7.2 h and 9.5 h, respectively. This means that, in relation to the mean, TTF distributions exhibit low variance, whereas the variance for TTR distributions is relatively large. However, note that time-scales of TTF and TTR times differ widely, as evidenced by the mean values.

The routing subsystem is fundamental to all communication in the Internet. With respect to Web Services, we expect routing failures to manifest as inability to communicate with a service, i.e. service unavailability.

Based on the study of Internet routes in [27] both Bernoulli and stochastic-process models of the routing system can be parameterised. The paper provides CDFs for the availability of routes, and CDFs for the MTTF and MTTR of route failures. All CDFs are over routes, i.e. specify the proportion of routes that exhibit a particular property. It is stated that the MTTF of the majority of routes is 25 days, with MTTR of at most 20 min.

2.2.4 Internet (Transport) We now consider transport in the Internet. As an abstract concept, the Internet itself is a single server whose service is data transfer, and where deviations could be both unavailability and delays of data transfer.

A two-state stochastic-process model for the availability of the Internet is presented in [28]. In the 'available' state, two hosts on the Internet are able to communicate, while in the 'unavailable' state the hosts cannot communicate. The model is parameterised based on several measurements in the period 1994–2000. The time-to-failure distribution is assumed to be exponential, with a mean of 48111 s. The time-to-repair distribution for unavailability events longer than 30 s[4] is approximated by the Pareto distribution with CDF $R(t) = 1 - 19t^{-0.85}$.* Truncated at 500,000 s, this distribution has a mean of 609 s. We note that the study in [16] (see Section 2.1.1) may also be employed in deriving Bernoulli and general random-variable models for Internet availability. However, in contrast to [28], in [16] the focus lies on host availability, not on network availability.

[FIXME: Re-check this.]

Let us now look at the protocols that the SOAP stack depends on. As the dominant SOAP transport in today's service-oriented systems, HTTP will be of interest. We then descend further in the SOAP stack (Figure 3) and investigate the TCP/IP level, which underlies the HTTP transport.

2.2.4.1 HTTP. When SOAP over HTTP is used without any further protocols (such as WSRM), SOAP may behave almost identical to HTTP, as far as its reliability and timeliness properties are concerned. There will be overhead due to SOAP processing, but SOAP does not add any reliability mechanisms.

Based on [30, 31, 32] general random variables can be used to describe response-times of HTTP connections. The study in [30] provides parameters for Pareto distributions that fit file-transmission times in the Web, noting that file sizes and network conditions only affect the location parameter, while the shape parameter is only affected by server load. The analyses in [31, 32] show that IP packet loss may have a strong impact on HTTP transmission times, furthermore, the response-times for subsequent attempts to the same location are often not correlated.

[4] Unavailability events shorter than 30 s are excluded, since they are considered irrelevant for the techniques studied in the paper ([29], p. 9).

To the best of our knowledge, no models that capture temporal behaviour of HTTP response times exist.* [FIXME: PR: to the best of MY knowledge...]

2.2.4.2 TCP/IP. Findings in the papers from the previous section illustrate that low-level failures affect quality of service, even though they are commonly masked by higher-level protocols, especially TCP, which provides reliable transport over links that exhibit IP packet loss. TCP's fault-handling mechanism for packet loss results in transmission delays, which then affect QoS. This section thus focusses on models for IP packet loss. Note that, while models for packet-loss may be difficult to introduce into abstract models of service-oriented systems, they can be easily employed in test-beds, where they allow the study of the complex timing-behaviours resulting from the interaction of the various fault-handling mechanisms in the SOAP and TCP/IP stack. From these, abstract models for higher-level behaviour can be obtained (cf. [7, 15]).

Bernoulli models for packet loss can be parameterised based on [33, 34, 35]. According to [33], 40% of the low-loss traces, but only 6% of the high-loss traces analysed can be described by Bernoulli packet loss. This may indicate that for high loss levels Bernoulli models are not appropriate. Only 1% of the traces analysed in [33] have loss rates larger than 10%. The authors of [34] remark that even in a LAN environment packet-loss rates may exceed 1%.* [FIXME: Put [35] in here somehow.]

Stochastic-process models for packet loss available in the literature may be divided into Markovian, i.e. variants of the Gilbert-Elliott model, and non-Markovian models. Both describe temporal dependencies in packet loss by at least two states with differing loss probability, and characterisations of the transitions between the states. Furthermore, one may distinguish between models that describe single-loss events (i.e. loss of one packet) and loss-episodes (i.e. consecutive loss of packets, or periods of elevated loss). The study of constancy of Internet properties in [33] concludes that packet-loss in change-free regions is well approximated by Markovian models: With low loss, 89% of traces are described by the two-state Gilbert model, 98% by a three-state Gilbert model, and 99% by four-state Gilbert models. For lossy traces, the numbers are 6%, 68%, 85%, and 96%. Furthermore, loss occurs in 'spikes' of consecutive losses, not in losses of nearby packets. This argues for models with loss probabilities of 0 and 1 in the loss-free and lossy states, respectively, and consequently a description of the loss process as interchanging loss-free periods and loss episodes. Loss-episode interarrivals are IID and well-modelled by exponential distributions at timescales up to 10 s. Beyond timescales of 10 s the authors observe correlation, which they attribute to a mixture of loss-processes at these timescales. The paper contains some plots of CDFs of inter-loss-episode- and loss-episode-lengths. Similar findings are presented in [36]. The analysis in [37] provides explicit parameters for three variations of the Gilbert model: Simple Gilbert (loss probability 0 and 1), Gilbert (loss probabilities 0 and $1-h$) and Gilbert-Elliot (loss probabilities $1-k$ and $1-h$). The models are fitted to data on multiple timescales. The data was obtained by feeding a trace into a bottleneck and observing packet loss due to tail drop.

	Bernoulli	Random Variable	Stochastic Process
Service	—	[10, 11, 12, 13, 14, 15]	—
Platform	[16]	—	[17, 18, 19, 16]
	—	—	—
Application Server/ SOAP Framework	—	—	[20, 21]
Virtualisation	—	—	[22]
Storage/Databases	[23, 24]	[24]	[3]
Communication	—	—	—
Web Services Stack	—	—	—
WS Infrastructure	—	—	—
WS Transport	—	[7]	—
Internet Infrastructure	[27, 25]	[26]	[27, 25]
Internet Transport	[33, 34, 35, 16]	—	[28, 30, 31, 33, 36, 37, 38, 39, 40, 41]

Table 1. Classification of fault-models for service-oriented systems by architectural level and fault-model type.

Non-Markovian models for packet loss can be characterised based on [38, 39, 40, 41]. In [38] interloss-times for non-congestion-related loss are described by a mixture of exponential distributions for intra-loss-burst times (with a length below 1 s), and a Normal distribution for the inter-loss-burst times (with mean 10 s). The authors propose a six-state Markov-Modulated Poisson Process (MMPP) to describe the inter-loss times. One state of the model is used to reproduce intra-loss behaviour. Unfortunately, the authors omit the parameters for their model. The studies in [39, 40] contain plots of CCDFs for the lengths of loss-free and lossy periods (regrettably omitting more detailed characterisations of the loss process, as the paper's focus is on a comparison of probe- and SNMP-based measurements). A Weibull distribution with shape parameter 2.9 and scale 0.3 is used to describe the loss-interarrival times in [41].

2.3 Concluding Remarks

Table 1 provides an overview of the papers containing fault-models reviewed in our study, classified by the components of a service-oriented system and the type of fault-model. We observe that the coverage by fault-models differs between the levels. For instance, while the Internet Transport level appears well-represented, there are no models for an abstract Communication between Services available. The degree of coverage also differs with respect to the types of models that are available. This is particularly regrettable for the Service level, whose only descriptions are general random variables (which are often not even specified very rigorously, cf. e.g. Section 2.1). Here, stochastic-process models that also describe changes in faults over time would be of great use, since fault-models of Web Services may be useful in answering many research questions related to service-oriented systems.

3 SOA Fault Classification Schemes

In assessing QoS aspects of service-oriented systems, fault-classification schemes serve three important purposes. First, they help to structure the study of available fault-models, second, they provide a structure for applying the fault-models in the system-model, and thus, third, a structured way for developing models* for SOA systems that are not limited to modelling a specific system, but instead represent general properties.

[FIXME: Should we call these performability models?]

We already applied a fault-classification scheme to structure our survey in the previous section. This scheme was inspired by architectural properties of SOA systems. Several other classification schemes for faults in service-oriented systems can be found in the literature. In this section we provide an overview of these. For each scheme, we will first summarise its important aspects. We then illustrate and discuss its use in classification of fault-models by applying it to the models surveyed in the first part of the paper. The classifications thus derived serve the second purpose identified above. An illustration of the application of classification schemes in developing general system-models, however, will be left for future work.

3.1 Phase/Level Classification

We begin with the classification scheme in [42], where faults are distinguished along a time dimension and an architectural dimension. In the time dimension, service-oriented systems are considered *phased-mission systems*, i.e. systems whose operational life consists of multiple phases with differing characteristics and requirements. The authors argue that this view is correct when considering the experience of a single user, where non-overlapping phases may be distinguished. Globally, however, phases tend to overlap. The following phases are considered typical: *Infrastructure discovery* and *client registration*, where the user discovers and enters the physical environment; *service registration*, *service discovery/lookup* and *service selection*, in which first the provider publishes the service, followed by the client discovering and then selecting and invoking the service; and *system configuration* and *service delivery*, comprised of the infrastructure and the service dynamically adapting to meet the client's requirements, and finally the delivery of the service to the client. In the architectural dimension, three levels are identified: The *end-point level*, which contains client- and server-side application objects; the *service delivery level*, comprising the entities concerned with delivering the service*; and the *lookup/discovery level*, consisting of the service registry and the discovery infrastructure. Additionally, the *network level* is mentioned, but not described in detail.

[FIXME: My interpretation]

We may attempt to use the classification scheme of [42] as a means to order the papers discussed in the survey comprising the first part of this paper. This then allows us to easily look up relevant papers for a given time-phase and architectural level. In order to do so, a few assumptions on our part are necessary. We interpret the End-point Level as equivalent to our high-level Web Service concept, the Delivery Level as the Platform and Storage levels, the

	End-Point Level	Service-Delivery Level	Lookup/Discovery Level	Network Level
Infrastructure Discovery	—	—	—	—
Client Registration	—	—	—	—
Service Registration	—	—	—	—
Service Discovery/Lookup	—	—	[10]	[26, 25]
Service Selection	—	—	—	—
System Configuration	—	—	—	—
Service Delivery	[10, 12, 13, 14, 15, 20]	[10, 12, 13, 14, 15, 20, 21, 22, 18, 17, 19, 23, 24, 3, 16]	[10]	[7, 26, 28, 29, 30, 31, 32, 33, 34, 35, 38, 39, 40, 41, 37, 25]

Table 2. Phase/Level Classification Scheme [42] applied to the studies in our survey.

Lookup/Discovery Level as the Web Services Infrastructure, and the Network level as a combination of our WS Stack and Internet categories.

In the Infrastructure-Discovery phase, faults in the DNS have a strong impact. These, analysed in [26], belong to the Network Level. For the Service-Discovery/Lookup phase, the analysis of UDDI entry validity in [10] provides a glimpse into faults at the Lookup/Discovery Level, while DNS faults ([26, 25]) again affect the Network Level. Turning our attention to the Service-Delivery phase, we note that the distinction between the End-Point and Service-Delivery levels becomes fuzzy, since the faults discussed in [10, 12, 13, 14, 15, 20] cannot be clearly attributed to either of the two. Assuming that the Service-Delivery Level is equivalent to our Platform and Storage levels, however, we can at least assign [21, 22, 18, 17, 19, 23, 24, 3] to the Service-Delivery Level. At the Lookup/Discovery Level, [10] again provides insight into UDDI faults. Finally, the studies in [7, 26, 28, 29, 30, 31, 32, 33, 34, 35, 38, 39, 40, 41, 37, 25], which are concerned with faults affecting communication, belong to the Network Level at the Service-Delivery phase. The resulting classification of the surveyed papers is shown in Tab. 2.

As pointed out in [42], the phase-level scheme has the advantage of capturing the fact that faults may have quite different effects, depending on when and where they occur. Note, however, that in Tab. 2 many rows are empty. This implies that at the respective phases none of the surveyed papers describe faults in any of architectural levels. On the other hand, some categories overlap, in that faults cannot be clearly assigned to either category. In fact, one might argue that many of the faults considered in the surveyed papers affect most or all of the time-phases and levels, since in each of the phases and at all levels the involved systems are often similar to each other. Database faults, for instance, may have

Service Side	[10, 12, 13, 14, 15]
Client Side Binding	—
Network and System	[20, 21, 22, 17, 18, 19, 23, 24, 3, 16, 7, 26, 28, 30, 31, 32, 33, 34, 35, 37, 39, 40, 38, 41, 25]

Table 3. Application of the Three-level Classification Scheme from [43].

an impact in every phase and at every level where storage and lookup of data is required; and IP packet loss affects all of the time-phases in service-oriented systems, since these systems rely on communication between their distributed components. While the overlap between categories may be partly due to a lack of rigour in the definition of the time-phases and architectural levels, it also appears likely that the scheme is too fine-grained to really describe real-world faults.

3.2 Three-level Classification

In [43] a scheme that classifies faults only along the architectural dimension is proposed. In contrast to [42], no time-dimension is considered. Three levels are distinguished: *Client-Side Binding*, i.e. faults occuring in the client during the binding (and perhaps also invocation) stage; *Network and System*, which comprises faults affecting the communication between the services, including faults in the Internet infrastructure and in the hosting subsystem (e.g. the application server); and *Service*, describing faults that occur in the service itself during processing.

Again, we employ this scheme to structure the papers in our survey, in order to be able to quickly identify studies concerned with faults of a given category. We map the levels in [42] to our categories as follows: The Service level is equivalent to our Web Service level, while the Network and System level comprises the remainder of the levels we distinguished. The Client-Side Binding level, however, does not correspond to any of our categories. This is due to the fact that it only describes programming faults, which have not been included in our survey. In fact, it appears that these faults might be better described as affecting the functional behaviour of the service. We obtain the classification shown in Tab. 3: [10, 12, 13, 14, 15] all belong to the Service-Side level, while the remainder are Network and System faults. Note that most papers belong to the third category. This is probably a drawback of the simplicity of the classification scheme, which only provides a very coarse granularity.

3.3 Time-Phase Classification

While the scheme in [43] only considers the architectural dimension, the taxonomy proposed in [44, 45] may be seen as focussing solely on the time-phases dimension of [42]. The taxonomy in [44, 45] is similar to [42], in that the authors identify five time-phases a service-oriented system has to traverse in order for the

service to be provided. Faults are classified according to the phases, as follows: The *Publishing* phase comprises faults originating during service-registration by the service provider. Faults occurring when the client discovers the service affect the *Discovery* phase. The *Composition* and *Binding* phases describe faults during service composition and binding to the required service; and the *Execution* phase contains all faults during execution of the service request. Each of the phases can be broken down further, and faults may be connected by the 'is-a' and 'causes a' relationships, thus allowing further refinement of the scheme.

In attempting to classify the fault-models surveyed in Section 2 to the five time-phases described in [44, 45] we first note that, due to the absence of an architectural dimension in the scheme, we cannot provide a mapping between the categories we used in our classification scheme and the time-phases of [44, 45]. Second, we observe that most of the faults considered in the refinements of the time-phases only affect the functional behaviour of the service. In fact, with the exception of the 'Timed Out' fault, *all* faults in the Publishing, Discovery, Composition, and Binding phases relate to functional behaviour. In the fifth phase (Execution), both the 'Timed Out' and 'Service Crashed' faults describe non-functional behaviour, while the 'Incorrect Result' fault again relates to functional behaviour. Furthermore, all faults defined for the Publishing phase relate to functional behaviour. With this in mind, we cannot assign any of the papers we surveyed to the first phase, while the remainder of the phases can be affected by any of the faults studied in the papers we considered, since each of these faults may result in 'Timed Out' faults. It thus appears likely that a notion of an architectural level is necessary in order to be able to meaningfully classify fault-models.

3.4 Multi-dimensional Classification

A multi-dimensional classification scheme is presented in [46]. The scheme is an application of the general concepts described in [5] to the field of service-oriented systems. The authors of [46] distinguish three broad categories of faults, and, orthogonal to these, three dimensions. The broad fault-categories are *Physical*, *Development* and *Interaction* faults, while the dimensions describe the phase of occurrence (during development/maintenance or during operation), the location where the fault occurs (internal or external to the system), and the architectural level (hardware or software).

Unfortunately, the definitions are lacking in clarity, recurring to examples instead. Nonetheless, based on the given examples, we may attempt the following mapping between the categories in our classification scheme and those in [46]: The Physical category comprises all levels below the Service and WS Stack. Development faults are faults on the Service level that affect functional behaviour, while non-functional behaviour is contained in the Interaction category. Then, we interpret the dimensions as follows: In the Phase dimension, all faults not directly and exclusively attributable to development or maintenance actions belong to the operational phase. The Internal category of the Boundary dimension contains all faults in our Service and WS Stack levels, while all faults

	Physical	Development	Interaction
Development	[24]	—	—
Operation	[20, 21, 22, 16, 17, 18, 19, 23, 24, 3, 25, 26, 27, 28, 30, 31, 32, 33, 34, 35, 37, 38, 39, 40, 41]	—	[10, 11, 12, 13, 14, 15, 7]
Internal	—	—	[10, 11, 12, 13, 14, 15, 7]
External	[20, 21, 22, 16, 17, 18, 19, 23, 24, 3, 25, 26, 27, 28, 30, 31, 32, 33, 34, 35, 37, 38, 39, 40, 41]	—	—
Hardware	[17, 23, 28, 30, 31, 32, 33, 34, 35, 37, 38, 39, 40, 41]	—	[7]
Software	[20, 21, 22, 16, 17, 18, 19, 23, 24, 3, 25, 26, 27, 28, 30, 31, 32, 33, 34, 35, 37, 38, 39, 40, 41]	—	[7]

Table 4. Multi-dimensional Classification based on [46].

below these levels are considered external. The Hardware/Software dimension does not directly map to any of our levels. We apply it by assigning fault-models that exclusively describe hardware faults to the former category, while all others belong to Software.

Table 4 displays the result of assigning fault-models from the surveyed papers to the categories of the multi-dimensional classification scheme of [46]. Note that the scheme (at least in our interpretation) is too coarse-grained to distinguish between most of the faults, as evidenced by the large number of papers that fall into the Physical category. Furthermore, the Development category is empty, since all studies considered here are only concerned with faults that affect non-functional behaviour. Finally, note that (with our interpretation) the Internal/External categories of the Boundary dimension is equivalent to the Physical/Interaction categories.

3.5 Fault-Model Ontology

The last approach we consider was proposed in [47]. This approach differs from the previous ones in that it provides a methodology, rather than a classification scheme. The methodology allows one to derive detailed descriptions of faults, with the ultimate goal of implementing test cases for these faults in a fault-injection testbed. In this approach, one starts with a coarse classification of

Physical Faults	[17, 3]
Software Faults	—
Resource Management Faults	—
Communication Faults	[7, 26, 27, 28, 30, 31, 32, 33, 34, 35, 37, 38, 39, 40, 41]
Lifecycle Faults	[23, 24, 3]

Table 5. An Application of the Fault-Model Ontology from [47].

faults, which is then successively refined until the faults of interest can be described sufficiently well for implementation in a test-bed. Thus the approach is similar to our concept of a fault-model, albeit with a focus on test-bed models.

The initial classification consists of five elements: *Physical faults*, *Software faults*, *Resource Management faults*, *Communication faults* and *Lifecycle faults*. Unfortunately, with the exception of Software faults (for which an example is given), the categories are not described in detail. Still, we will try to apply them to our survey. In doing so, we assume that the Physical faults category corresponds to faults that can be uniquely identified as originating in hardware, and Lifecycle faults refer to operator faults. Table 5 presents the resulting classification. Note that the Software and Resource Management categories are empty; furthermore, many of the reviewed papers are missing from the table, since we could not identify an appropriate category.

3.6 Concluding Remarks on Fault Classification Schemes

Several fault-classification schemes for service-oriented systems have been proposed in the literature. The phase-level scheme in [42] considers both a time- and an architecture-dimension, thereby accounting for the fact that the effects of faults may differ, depending which phase they affect. The three-level scheme in [43] focusses solely on the architecture dimension, while the taxonomy in [44, 45] instead considers only the time-dimension. The multi-dimensional scheme in [46] also considers architectural and time dimensions, splitting the former by referring to the system boundary and the difference between hardware and software faults. Furthermore, the scheme in [46] adds a distinction between functional and non-functional behaviour. Finally, the ontology in [47] provides a mixture of an architectural dimension and an abstract concept of faults. Note, however, that neither [46] nor [47] contain sufficiently strict definitions of the respective categories, and thus our discussion of these approaches is based on our interpretation.

A number of observations can be made from our application of these approaches to the classification of the fault-models surveyed in Section 2. First, it appears that a purely time-phase-based taxonomy may not be sufficient to distinguish real-world faults. This implies that some notion of system architecture is required. Second, one should be careful to choose the right granularity for the classification of faults. When categories are too fine-grained, as in [42], many categories might not relate to observable faults. On the other hand, if the

categories are too coarse (as in [43] and [46]), distinction between faults becomes difficult, since many faults are lumped into the same category. Third, strict and correct definition of fault categories is important. This is highlighted by the approaches in [46] and [47], which required a large amount of interpretation before they could be applied to the surveyed studies.

We conclude this section by remarking that for the purposes of fault-model classification the recursive approach we applied in Section 2 appears the most promising, even though it does not consider the time-dimension. If a time-dimension is required, the scheme in [42], or a combination of this scheme with our scheme, may be most applicable. In cases where a coarser and purely architectural taxonomy suffices, one may apply the one from [43]. The schemes proposed in [46] and [44, 45], however, do not appear well-suited to the classification of real-world faults.

4 Conclusion

In this paper we have surveyed fault-models for use in system-models of service-oriented systems and presented an overview of fault-classification schemes for faults in SOA systems. In the first part we observed that the description of faults in many components of typical SOA systems is lacking both in the number of studies and in the detail of the fault-models, since many authors just analyse statistical properties of the data and do not provide explicit fault-models. The second part showed that fault-classification schemes often exhibit weaknesses in their actual application, which manifest as both overlapping and empty categories.

We hope that both parts of the survey will help researchers wishing to improve understanding of faults in service-oriented systems. The survey of fault-models will hopefully serve as a guide to direct research efforts to those areas still poorly understood, and perhaps also improve the value of analyses by underlining the importance of detailed fault-models for further research. One potential application of fault-classification schemes will be addressed in a follow-up study, where we illustrate and evaluate their use for structuring the study of QoS in service-oriented systems both in test-bed and abstract system models.

Acknowledgements

We thank Aad van Moorsel for his helpful comments on this paper. This work was supported by DFG grants Wo 898/2-1 and Wo 898/3-1.

References

[1] Neuts, M.F.: Matrix-Geometric Solutions in Stochastic Models. An Algorithmic Approach. Dover Publications, Inc., New York (1981)

[2] van Moorsel, A., Bondavalli, A., Pinter, G., Madeira, H., Majzik, I., Durães, J., Karlsson, J., Falai, L., Strigini, L., Vieira, M., Vadursi, M., Lollini, P., Esposito, R.: State of the art. Technical Report D2.1, Assessing, Measuring, and Benchmarking Resilience (AMBER) (April 2008)
[3] Vieira, M., Madeira, H.: Definition of Faultloads Based on Operator Faults for DMBS Recovery Benchmarking. In: PRDC '02: Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing, Washington, DC, USA, IEEE Computer Society (2002) 265
[4] Cristian, F.: Understanding fault-tolerant distributed systems. Communications of the ACM **34** (1991) 56–78
[5] Avižienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing **1**(1) (2004) 11–33
[6] Hohlfeld, O., Geib, R., Haß linger, G.: Packet Loss in Real-Time Services: Markovian Models Generating QoE Impairments. In: Proc. of the 16th International Workshop on Quality of Service (IWQoS). (June 2008) 239–248
[7] Reinecke, P., Wolter, K.: Phase-Type Approximations for Message Transmission Times in Web Services Reliable Messaging. In Kounev, S., Gorton, I., Sachs, K., eds.: Performance Evaluation – Metrics, Models and Benchmarks. Volume 5119 of Lecture Notes in Computer Science., Springer (June 2008) 191–207
[8] BEA Systems, IBM, Microsoft Corporation Inc, TIBCO Software, Inc.: Web Services Reliable Messaging Protocol (WS-ReliableMessaging) (February 2005)
[9] Sun Microsystems: Java adventure builder reference application. https://adventurebuilder.dev.java.net/ (2006) Last seen April 28th, 2009.
[10] Kim, S.M., Rosu, M.C.: A survey of public web services. In: WWW Alt. '04: Proceedings of the 13th international World Wide Web conference (Alternate track papers & posters), New York, NY, USA, ACM (2004) 312–313
[11] Gorbenko, A., Kharchenko, V., Tarasyuk, O., Chen, Y., Romanovsky, A.: The threat of uncertainty in service-oriented architecture. Technical Report 1122, Newcastle University, School of Computing Science (Oct 2008)
[12] Datla, V., Goseva-Popstojanova, K.: Measurement-based performance analysis of e-commerce applications with web services components. In: Proc. IEEE International Conference on e-Business Engineering ICEBE 2005. (2005) 305–314
[13] Juse, K.S., Kounev, S., Buchmann, A.: PetStore-WS: Measuring the Performance Implications of Web Services. In: Proceedings of the 29th International Conference of the Computer Measurement Group on Resource Management and Performance Evaluation of Enterprise Computing Systems (CMG 2003), Dallas, Texas, USA, December 7-12, 2003, Computer Measurement Group (CMG) (December 2003) 113–123
[14] Tian, M., Voigt, T., Naumowicz, T., Ritter, H., Schiller, J.: Performance impact of web services on internet servers. In: Proceedings of IASTED International Conference on Parallel and Distributed Computing and Systems, Marina Del Rey, California, USA (2003)
[15] Reinecke, P., Wittkowski, S., Wolter, K.: Response-time Measurements Using the Sun Java Adventure Builder. In: QUASOSS '09: Proceedings of the 1st International Workshop on Quality of Service-oriented Software Systems, New York, NY, USA, ACM (2009) 11–18
[16] Merzbacher, M., Patterson, D.: Measuring end-user availability on the web: Practical experience. In: DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks, Washington, DC, USA, IEEE Computer Society (2002) 473–477

[17] Schroeder, B., Gibson, G.A.: A large-scale study of failures in high-performance computing systems. In: DSN '06: Proceedings of the International Conference on Dependable Systems and Networks, Washington, DC, USA, IEEE Computer Society (2006) 249–258
[18] Oppenheimer, D., Ganapathi, A., Patterson, D.A.: Why do internet services fail, and what can be done about it? In: USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems, Berkeley, CA, USA, USENIX Association (2003) 1–1
[19] Long, D., Muir, A., Golding, R.: A longitudinal survey of internet host reliability. In: SRDS '95: Proceedings of the 14th Symposium on Reliable Distributed Systems, Washington, DC, USA, IEEE Computer Society (1995) 2
[20] Tang, D., Kumar, D., Duvur, S., Torbjornsen, O.: Availability measurement and modeling for an application server. In: DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks, Washington, DC, USA, IEEE Computer Society (2004) 669
[21] Silva, L., Madeira, H., Silva, J.: Software aging and rejuvenation in a soap-based server. In: Proc. Fifth IEEE International Symposium on Network Computing and Applications NCA 2006. (2006) 56–65
[22] Kourai, K., Chiba, S.: A fast rejuvenation technique for server consolidation with virtual machines. In: DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Washington, DC, USA, IEEE Computer Society (2007) 245–255
[23] Costa, D., Rilho, T., Madeira, H.: Joint evaluation of performance and robustness of a cots dbms through fault-injection. In: Proc. International Conference on Dependable Systems and Networks DSN 2000. (2000) 251–260
[24] Vieira, M., Madeira, H.: A dependability benchmark for oltp application environments. In: VLDB '2003: Proceedings of the 29th international conference on Very large data bases, VLDB Endowment (2003) 742–753
[25] Pang, J., Hendricks, J., Akella, A., De Prisco, R., Maggs, B., Seshan, S.: Availability, usage, and deployment characteristics of the domain name system. In: IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, New York, NY, USA, ACM (2004) 1–14
[26] Pappas, V., Xu, Z., Lu, S., Massey, D., Terzis, A., Zhang, L.: Impact of configuration errors on dns robustness. In: SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM (2004) 319–330
[27] Labovitz, C., Ahuja, A., Jahanian, F.: Experimental study of internet stability and backbone failures. In: FTCS '99: Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, Washington, DC, USA, IEEE Computer Society (1999) 278
[28] Dahlin, M., Chandra, B.B.V., Gao, L., Nayate, A.: End-to-end wan service availability. IEEE/ACM Trans. Netw. **11**(2) (2003) 300–313
[29] Dahlin, M., Chandra, B.B.V., Gao, L., Nayate, A.: End-to-end wan service availability (extended version). Technical report, University of Texas at Austin (2003)
[30] Nossenson, R., Attiya, H.: The distribution of file transmission duration in the web: Research articles. Int. J. Commun. Syst. **17**(5) (2004) 407–419
[31] Reinecke, P., van Moorsel, A.P.A., Wolter, K.: A measurement study of the interplay between application level restart and transport protocol. In Malek, M., Reitenspieß, M., Kaiser, J., eds.: Service Availability. Proceedings of the First International Service Availability Symposium. Volume 3335 of LNCS., Springer (May 2004) 86–100

[32] Reinecke, P., van Moorsel, A.P.A., Wolter, K.: Experimental Analysis of the Correlation of HTTP GET Invocations. In Horváth, A., Telek, M., eds.: Formal Methods and Stochastic Models for Performance Evaluation. Volume 4054 of LNCS., Springer (June 2006) 226–237
[33] Zhang, Y., Duffield, N., Paxson, V., Shenker, S.: On the constancy of internet path properties. In: IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, New York, NY, USA, ACM (2001) 197–211
[34] Pang, R., Allman, M., Bennett, M., Lee, J., Paxson, V., Tierney, B.: A first look at modern enterprise traffic. In: IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement, Berkeley, CA, USA, USENIX Association (2005) 2–2
[35] Chung, S.H., Won, Y.J., Agrawal, D., Hong, S.C., Ju, H.T., Park, K.: Detection and analysis of packet loss on underutilized enterprise network links. In: E2EMON '05: Proceedings of the End-to-End Monitoring Techniques and Services on 2005. Workshop, Washington, DC, USA, IEEE Computer Society (2005) 164–176
[36] Yajnik, M., Moon, S., Kurose, J., Towsley, D.: Measurement and modelling of the temporal dependence in packet loss. In: Proc. IEEE Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM '99. Volume 1. (1999) 345–352
[37] Haßlinger, G., Hohlfeld, O.: The gilbert-elliott model for packet loss in real time services on the internet. In Bause, F., Buchholz, P., eds.: MMB 2008, VDE Verlag (2008) 269–286
[38] Hacker, T., Noble, B., Athey, B.: The effects of systemic packet loss on aggregate tcp flows. In: Proc. ACM/IEEE 2002 Conference Supercomputing. (2002) 7–7
[39] Barford, P., Sommers, J.: A comparison of probe-based and router-based methods for measuring packet loss. Technical report, University of Wisconsin-Madison (September 2003)
[40] Barford, P., Sommers, J.: Comparing probe- and router-based packet-loss measurement. IEEE Internet Computing **8**(5) (2004) 50–56
[41] Hacker, T., Smith, P.: Building a network simulation model of the teragrid network. In: TeraGrid 2008 Conference, Las Vegas, NV (June 9-13 2008)
[42] Cotroneo, D., Flora, C., Russo, S.: Improving dependability of service oriented architectures for pervasive computing. WORDS **00** (2003) 74
[43] Gorbenko, A., Mikhaylichenko, A., Kharchenko, V., Romanovsky, A.: Experimenting with exception handling mechanisms of web services implemented using different development kits. Technical Report 1010, Newcastle University, School of Computing Science (March 2007)
[44] Brüning, S., Weißleder, S., Malek, M.: A fault taxonomy for service-oriented architecture. Technical Report 215, Humboldt-Universität zu Berlin (2007) [Online: Stand 2008-10-17T07:41:39Z].
[45] Brüning, S., Weißleder, S., Malek, M.: A fault taxonomy for service-oriented architecture. In: IEEE International Symposium on High-Assurance Systems Engineering, Los Alamitos, CA, USA, IEEE Computer Society (2007) 367–368
[46] Chan, K.S.M., Bishop, J., Steyn, J., Baresi, L., Guinea, S.: A fault taxonomy for web service composition. In: in Proceedings of the 3 rd International Workshop on Engineering Service Oriented Applications (WESOA07), Springer LNCS. (2007)
[47] Looker, N., Xu, J., Munro, M.: Determining the dependability of service-oriented architectures. International Journal of Simulation and Process Modelling **3**(1/2) (2007) 88–97