Modeling Service Composition Reliability in Pervasive Computing

Klaus Marius Hansen, Jeppe Brønsted

RH-02-2010

Science Institute
University of Iceland
Dunhaga 3, 107 Reykjavík

kmh@hi.is

March 2010

Report nr. RH-02-2010, Reykjavík 2010

Klaus Marius Hansen, Jeppe Brønsted.
Modeling Service Composition Reliability in Pervasive Computing,
Science Institute, University of Iceland, Technical report RH-02-2010, March 2010

Science Institute, University of Iceland, Dunhaga 3, IS-107 Reykjavík, Iceland

Abstract

In pervasive computing, new functionality is often realized by recombining existing functionality in the form of service compositions. As pervasive environments inhabit ever greater parts of our lifes, it becomes important to which degree we can justifiably trust service compositions. In this report, we address an important component of trust, namely reliability of service compositions. In particular, we present an approach to model and estimate the reliability of service compositions in terms of reliability of services and implement the approach in an existing, REST-based service framework for pervasive computing. Our evaluations through implementation and simulation show that our approach can model useful service composition problems, is implementable on resource-constrained devices, and is sufficiently accurate in reliability estimation.

Contents

Chapter 1

Introduction

One of the seven challenges for ubiquitous computing in the home listed by Edwards and Grinter [11] is *reliability* (i.e., the continuity of a correct service). They point out that in the home, unlike in a desktop environment, the expectations with respect to reliability are very high. We do not expect to upgrade the software on our microwave oven and we expect light to turn of when a switch is pushed. Given that a significant part of the functionality in the smart home of the future will be realized by means of service composition, an important part of the home owners living experience will be influenced by the reliability of these compositions.

The reliability of appliances in the home is relatively static throughout the lifetime of the products, but composite services, depending on a possibly varying set of composed services, will have dynamic reliability depending on the services currently available. If the reliability of a service, e.g., depends on mobile devices, the service will only be available when the mobile device is reachable.

More generally, we may consider *dependability* in the context of pervasive computing. Dependability may be defined as the ability of a system to deliver service that can justifiably be trusted or alternatively the ability to avoid service *failures* that are more frequent and more severe than is acceptable [20, 2]. Dependability is an integrating concept that covers the non-functional properties of availability (i.e., the readiness of a correct service), reliability, safety (i.e., the absence of catastrophic consequences on user and the environment), integrity (i.e., the absence of improper system alterations), and maintainability (i.e., the ability to undergo modifications and repairs) [2].

Failures occur if a *fault* in a system is allowed to manifest itself as an *error* triggering a transition of a service from a correct service to an incorrect service. Thus to build dependable systems, faults should either be *prevented* from being introduced, be *removed* if introduced, or *tolerated* when the system is in operation if not removed. In addition, fault *forecasting* may qualitatively or quantitatively help in assessing if service failures are sufficiently infrequent or have acceptable severity. In this report, we focus on fault forecasting, quantitatively estimating the availability of service

compositions.

While all aspects of dependability are important, we are here concerned with reliability and availability (of service compositions). A key factor in improving the reliability of service composites, is the ability to estimate the reliability of a particular composite. A computed reliability can be used, e.g., at design time to get hints to whether the composite will perform satisfactorily or at runtime to notify a user about changing reliabilities and to dynamically adapt the composite to changing reliability conditions. Several challenges for an approach to reliability estimation of service composition in pervasive computing must be resolved: The approach should be

1. *Applicable* in that it should be able to model and estimate the reliability of useful and realistic service compositions
2. *Accurate* in that estimated reliabilities should correspond to actual reliabilities
3. *Realizable* in that reliability modeling and estimation at design time and runtime should be implementable in a pervasive computing context including on heterogeneous, resource-constrained devices

Our main contribution in this report is an approach (consisting of a theoretical model and a practical implementation) to reliability estimation of service composites in pervasive computing that addresses the challenges.

In the following, we will return to our three challenges and how we address them, first by defining two service composition scenarios that will be used as examples and as the basis of our evaluation.

Chapter 2

Service Composition Reliability Scenarios

We chose to use scenarios as context for the description and evaluation of our approach, because they underline the connection between theory and practice and help ensure that models and implementation can support relevant and realistic situations. Furthermore, they provide a concrete understandable setting that makes it easier to convey complex ideas.

In the following we describe two scenarios. The first, a simple power save scenario, contains basic requirements for service composition and reliability estimation and is used to demonstrate that out approach is *applicable* and *realizable*. In the second scenario, peak demand response, a set of collected measurements allows us to reason about the *accuracy* of our approach. Additionally, this scenario also gives rise to a more subtle notion of reliability.

2.1 Saving power

Consider the following scenario:

> The Hansens have been living in a smart home for several years. All lighting in the house is digitally monitored and controlled and wireless motion sensors have been installed in each room to detect presence. Typically, a single motion sensor will not cover a whole room and therefore each room has more than one sensor. In addition, some motion sensors cover areas in multiple rooms. To save energy, a composite service for each room receives input from the sensors and turns off the light in the room when nobody is present. When the sensors run out of battery, the reliability of the power saving composites decrease, but every month the inhabitants receive a status of all composites in the home with reliability

information, allowing the father to change batteries in the sensors when the reliability gets too low.

In the scenario above, the availability of reliability data is important because the functionality of the composite is only directly observable when nobody is present. Without some monitoring of the composite service, the amount of saved power could drop without anybody noticing (until the electricity bill arrives).

In figure 2.1, an instance of the scenario is illustrated by means of a Boolean circuit diagram. Even though a single motion sensor fails, the composite will still continue to function. Thus, the composite p is considered functioning if the lamp l_1 is functioning and at least one of the motion sensors s_1 and s_2 is functioning. This relationship is described using "and" and "or" gates in the figure.

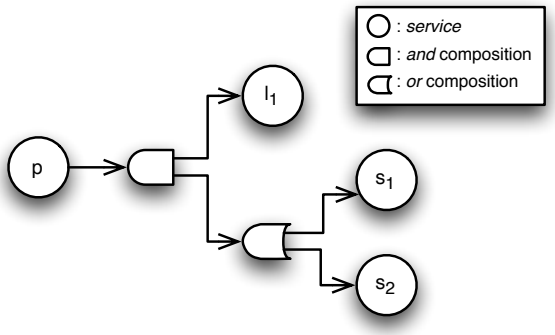

Figure 2.1: Power Save scenario

2.2 Peak demand response

In electricity grids, the amount of electricity generated should roughly equate the amount consumed to maintain the stability of the grid. Generation and transmission systems are therefore dimensioned to accommodate the total peak demand plus some error margin. When the demand approaches the capacity of the grid, then risk of potential disturbances increases. For electrical utility companies, it is therefore highly desirable to be able to intelligently control power consumption during peak load. Hereby, the company can save capital by building smaller plants and can minimize the risk of electrical disturbances, such as e.g. brownouts. Demand response can be achieved by cutting power to a selected set of devices of less importance, such as e.g. air conditioning or floor heating. The perceived service quality by the customers is significantly higher, compared to brownouts, and the utility company can operate closer to its maximum capacity.

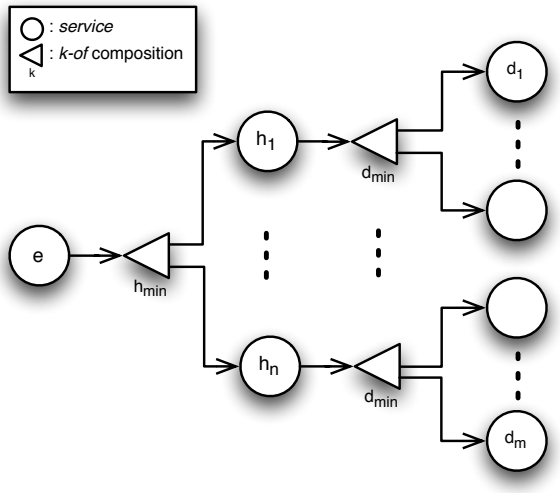

Figure 2.2: Peak Demand Response scenario

Figure 2.2 shows an emergency demand response scenario composition. In the scenario, the electrical utility company implements a service, e, that allows turning off appliances in houses based on agreements with house owners. In the scenario, the utility company has agreements with n houses and the houses in total have m devices available for the peak demand response service.

For e to be available, at least h_{min} houses each having at least d_{min} devices available to be turned off are needed. Each house runs a service, h_i, that e contacts to turn off appliances while the house services, h_i, contacts device services d_j that are deployed in the house. Thus, availability in this scenario is concerned with how much the electrical utility company can lower power usage (during peak demand).

Chapter 3

Related Work

Service composition has been widely researched both in distributed computing in general and in the context of pervasive computing in specific [24, 6]. While our main contributions of this report are related to reliability prediction of service compositions, the eventual goal of our service composition mechanism is to enable an equivalent of mash-ups for embedded services in the home. A technical consequence of this is that we build on an architecture in which devices are service-enabled based in an HTTP-based REpresentational State Transfer (REST; [13]) style. Approaches exists for this for location-aware service [5] and for SOAP-based services [22]. While such a composition mechanism can potentially be easily applied to produce innovative compositions, the quality of such a composition may be problematic [17]. One goal of our research is to provide a service reliability prediction mechanism that can be usable in this context.

In service-oriented computing, in particular the reliability, integrity, and maintainability attributes of dependability have been widely researched for single services. Fault removal and fault forecasting are, e.g., typically aided through using fault injection to quantify dependability [20, 10], fault tolerance of services are often supported by middleware abstractions such as the CORBA Transaction Service [9], and model-driven techniques are often used to prevent fault introduction [32]. Reliability of service compositions has received less attention and our work relies on reliability estimation of individual services to evaluate reliability of composed services. Particular issues in service composition for pervasive computing include that compositions need to support context awareness, manage contingencies, leverage device heterogeneity, and eventually empower the end-users [6]. In this report, we focus in particular on support for managing contingencies while allowing heterogeneous devices to participate in compositions.

Brønsted et al. [6] recently analyzed service composition issues in pervasive and ubiquitous computing. Of 24 investigated service composition mechanisms, seven considered Quality of Service (QoS) requirements [1, 4, 16, 18, 25, 30, 31]. Of these

neither Baresi et al. [4], Sousa et al. [30], nor Amundsen and Eliassen [1] consider reliability explicitly. Issarny et al. [16], Yang et al. [31], and Kalapasur et al. [18] models reliability to some extent, although their approaches are not described in detail. Finally, Mokthar et al. [25] consider service composition in a Semantic Web context, using OWL-S [23] in an Ambient Intelligence setting. Compositions are modelled as finite state machines and uses the approach of Cardoso et al. [8]. This approach iteratively reduces compositions to a single, virtual service having the QoS of the composition. The state machines thus essentially implicitly describes the availability requirements of a composition whereas our approach allows for an explicit representation. Furthermore, we provide and evaluate a lightweight implementation of reliability/availability estimation.

Ubiquitous and pervasive computing applications are moving into domains in which dependability (among which reliability is a component) is highly important. Bardram and Nørskov recently considered context awareness in the safety-critical context of an operating room [3]. Bardram and Nørskov argue that the degree of *confidence* in the system needed is what distinguishes safety-critical context-aware systems from other context-aware systems. While reliability certainly does not imply safety [21], one concept of confidence (or "trust" as in definitions of dependability) is reliability and availability. Here our approach could eventually play a role in evaluating the confidence that can be put in service compositions.

Chapter 4

Modeling Service Composition Reliability

Central concepts of dependability are are reliability and availability. In the context of service composition, we will talk interchangeably about reliability and availability and define it as the probability that an item functions correctly at time t [26]. In terms of service (composition), being available then means that the service is accessible and able to compute a correct function at time t.

Often, service compositions require that all services in the composition are available at the time of invocation for the composed service to be available. An example of this is a composite service that turns of all lights in a home using a "master switch". Here all lights are required to function as well as the switch in order for the composition to be available. We model this as an "and" form of reliability composition.

Furthermore, to increase reliability, redundant services are often introduced. An example of this is the Power Save scenario in which two light sensors redundantly provide the same reliability. We model this as an "or" form of reliability composition.

Both "and" and "or" composition can be generalized as "k-out-of-n" compositions in which k out of n services are required to function for the composition to function. Here, an "and" composition is an "n-out-of-n" composition and an "or" composition is a "1-out-of-n" composition.

We model "and", "or", and "k-out-of-n" reliability of service compositions using techniques derived from *Reliability Block Diagrams* [26]. Specifically, we are concerned with the *state* of each service, s, at time, t, and model this as a binary stochastic process:

$$X_s(t) = \begin{cases} 1 & \text{if } s \text{ is functioning at time } t \\ 0 & \text{otherwise} \end{cases}$$

We call such services *simple*. The definition of "functioning" is application-specific: in the power save scenario, a "functioning" service is one that can be accessed and

that delivers a correct result. In the energy demand response scenario, a functioning device "functioning" is one that is consuming power and can be switched off.

For a composite, c, we furthermore define its structure function as:

$$\phi_c(t) = \begin{cases} 1 & \text{if } c \text{ is functioning at time } t \\ 0 & \text{otherwise} \end{cases}$$

As an example, take an "and" composition of two services, s_1 and s_2. For this composition, the structure function is:

$$\phi_c(t) = X_{s_1}(t) \cdot X_{s_2}(t)$$

since the structure is functioning if and only if both of s_1 and s_2 are functioning. For the general k-out-of-n case, a composition c' of $s_1, ..., s_n$ (where an s_i may be composite service itself) is functioning if at least k out of its n composed services are functioning. In the next section, we present an algorithm to compute the structure function in terms of states of the simple services of a composition.

The availability of a service composition c may now be expressed as:

$$A_c(t) = Pr(\phi_c(t) = 1)$$

Thus a key to estimating availability of a service composition is to calculate the structure function of the service composition. We turn to this next.

4.1 Calculating Structure Functions

To calculate the reliability (average availability) of a service composition, s, we first calculate the structure function for the service composition using the following recursive algorithm based on Rushdi's algorithm [28, 29]. (In the following, we drop references to time for convenience.)

1. If s is a simple service, then the structure functions of the service is

   $$\phi(s) = X_s$$

   where X_s is the state of service s.

2. If s is a k-out-of-n composite, the structure function, $\phi(of(s_1, ..., s_n; k))$ of s is

   $$0, \quad \text{if } k > n \tag{4.1}$$
   $$1, \quad \text{if } k = 0 \tag{4.2}$$
   $$\phi(s_n) \cdot \phi(of(s_1, ..., s_{n-1}; k-1)) + (1 - \phi(s_n)) \cdot \phi(of(s_1, ..., s_{n-1}; k)), \quad \text{otherwise} \tag{4.3}$$

Special cases of this is the structure function of an "and" composition of n services:

$$\phi(of(s_1, ..., s_n; n)) = \prod_i^n \phi(s_i) \tag{4.4}$$

and of an "or" composition of n services:

$$\phi(of(s_1, ..., s_n; 1)) = 1 - \prod_i^n (1 - \phi(s_i)) \tag{4.5}$$

Observe that since the state variables are binary $X_s^2 = X_s$ for any simple service s. This means that we may reduce ϕ to linear form:

$$\phi(s) = \sum_{i=1}^{2^n} m_i \cdot ind(i, 1) \cdot X_{s_1} \cdot \cdots \cdot ind(i, n) \cdot X_{s_n} \tag{4.6}$$

where $m_i \in \mathbb{N}$ and ind is an indicator function

$$ind : \{1, ..., 2^n\} \times \{1, ..., n\} \mapsto \{-1, 0, 1\}$$

While the expression in equation (4.6) can typically be (much) reduced, it may in general have 2^n terms, i.e., the time and space complexity of the calculation of the structure function (and subsequently of availability) is worst case exponential.

4.2 Calculating Availability

We observe that since the structure function (at time t), ϕ, for a service, s, is a binary random variable, the expectation value is:

$$\begin{aligned} E[\phi(s)] &= 0 \cdot Pr(\phi(s) = 0) + 1 \cdot Pr(\phi(s) = 1) \\ &= Pr(\phi(s) = 1) \\ &= A_s \end{aligned}$$

Thus, to calculate the reliability/average availability of s, we need to calculate $E[\phi(s)]$. For random variables X and Y and constants a and b, we generally have:

$$E[aX + bY] = aE[X] + bE[Y] \tag{4.7}$$

Furthermore, for two binary and independent random variables, we have:

$$\begin{aligned} E[X \cdot Y] &= 0 \cdot Pr(X \cdot Y = 0) + 1 \cdot Pr(X \cdot Y = 1) \\ &= Pr(X = 1) \cdot Pr(Y = 1) \\ &= E[X] \cdot E[Y] \end{aligned} \tag{4.8}$$

Assume now that the states of the simple services of a composition are independent. Then, combining equations (4.6), (4.7), and (4.8), we may calculate the availability of the composite service as a function of the availability of its constituent simple services:

$$A_s = \sum_{i=1}^{2^n} m_i \cdot ind(i,1) \cdot A_{s_1} \cdot \cdots \cdot ind(i,n) \cdot A_{s_n} \tag{4.9}$$

Note that it is not in general possible to calculate availability in a simple, recursive way (following the structure of our algorithm for structure function calculation) since the same service may be part of more than one (sub-)composition. Since we may need to store $O(2^n)$ terms to compute (4.9), it is interesting (in particular in a pervasive, resource-constrained setting) to consider how less space can be used. We consider this in the next section.

4.3 Space-Optimized Availability Computation

In many typical cases, service compositions will primarily make use of the “and" and “or" constructs, resulting in a simple structure function, cf. equations (4.4) and (4.5), and thus in simple availability calculation.

To consider when we need less storage (of terms of (4.6)) for $\phi(of(s_1,...,s_n;k))$, we observe that if $s_1,...,s_n$ are simple and distinct then we may calculate the availability directly as

$$0, \quad \text{if } k > n \tag{4.10}$$

$$1, \quad \text{if } k = 0 \tag{4.11}$$

$$A_{s_n} \cdot A_{of(s_1,...,s_{n-1};k-1)} + (1 - A_{s_n}) \cdot A_{of(s_1,...,s_{n-1};k)}, \quad \text{otherwise} \tag{4.12}$$

This holds since any s_i do not reappear in the recursive calculation of $\phi(of(s_1,...,s_{i-1});k-1)$ and $\phi(of(s_1,...,s_{i-1});k)$. In general, if a service s_i does not compose another service s_j more than once and if s_i is not composed more than once by another service, this holds.

In our implementation (see later) this concretely means that if a service s has this property, we do not need to linearize (nor calculate) the structure function expression for s, but can calculate its availability at runtime based on the availability of its composed services using (4.12).

Chapter 5

Implementing Service Reliability Models

The techniques for computation of reliability of composite services have been implemented in the Homeport infrastructure [7]. The Homeport infrastructure is a service infrastructure designed for the integration of home automation devices using heterogeneous communication protocols. The integration strategy is not to enforce a unifying protocol but rather to enable co-existence of wired and wireless communication standards and protocols. This is motivated by the need for business to protect investments and remain in control of subsystems. Integration is realized by letting a set of gateways - homeports - present subsystem functionality through a standardized service protocol. This is illustrated in figure 5.1, where homeports enable access to, e.g., ZigBee and Zwave devices. The architecture is completely decentralized in that no particular homeport acts as a governing entity. Hereby, subsystems can be added at runtime without affecting existing services and applications.

The Homeport gateway software is implemented in Python and uses HTTP and the principles of REST to expose services. The gateway software runs on the relatively resource constrained platform, Linksys NSLU2. The NSLU2 has a 266 MHz ARM Intel XScale processor, consumes approximately 5 W of power, and runs the limited SlugOS Linux distribution.

The support for determining reliability of composite services consists of three parts. Firstly, an XML format for describing the structure of composites allows the developer of a composition mechanism to declare, automatically or manually, how a composite depends on its constituent services. Secondly, the reliability approximator (RA) use multiple methods for estimating reliability of individual, non-composite services. Finally, the composite reliability estimator (CRE) use this information at composition-time and/or runtime to estimate the reliability of composite services. In the following, we describe each part in detail.

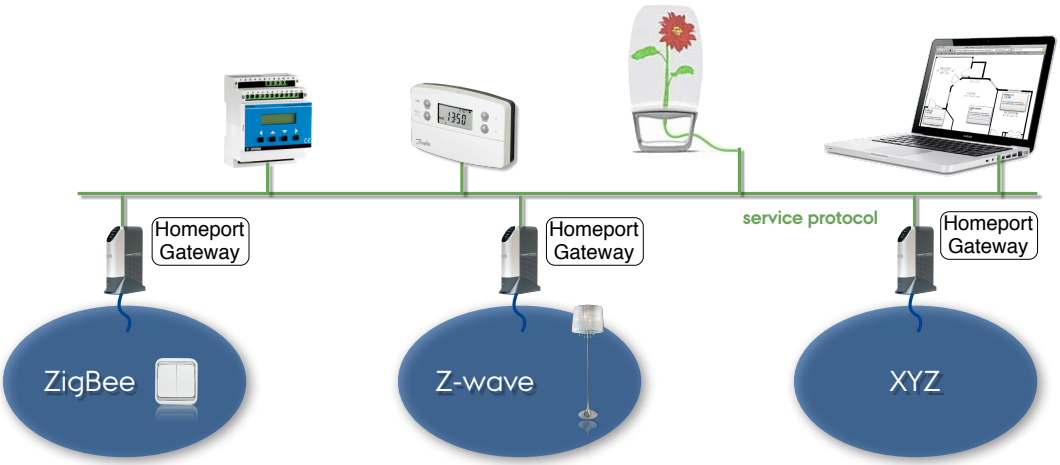

Figure 5.1: Homeport deployment

5.1 Composition Structure Description Format

The composition structure description format (CSDF) is an XML format that allows the composite reliability estimator to estimate the reliability of composite services by specifying how the reliability of the composite depends on the reliability of the services it is composed from. Additionally, the XML format acts as an interface between the abstract reliability estimation algorithm and the heuristics that determine reliability of individual services. CSDF documents can be generated from service composite specifications such as e.g. Homeport Composition Language [7] or can be written by hand to describe hardware controller logic.

In listing 5.1, an example of a CSDF document that describes the structure of a composite service that implements the 'Saving Power' scenario is shown. The document describes the same structure as the one shown in figure 2.1.

Each composite in the specification contains a `<requiredservices>` element that is a tree structure containing node elements `<and>`, `<or>`, and `<kofn>`, and leaf elements `<service>`. The `<kofn>` element `<kofn required="2" >...</kofn>` denotes that the composite is considered available only if at least 2 of the sub elements are available. `<or>` and `<and>` are aliases for `<kofn required="1" >...</kofn>` and `<kofn required="all" >...</kofn>`, respectively.[1]

For the CSDF document in listing 5.1, the `lamp` service is required and, in addition, at least one of `ultrasound-motion-sensor` and `infrared-motion-sensor` has to be available. The CRE uses the `<requiredservices>` element to estimate the reliability of the composite.

[1]The XML schema for CSDF can be seen in Appendix A

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <compositelist xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:noNamespaceSchemaLocation="csdf.xsd">
4    <composite
5      name="simple-power-save" id="cmp-1234"
6      language="python" src="NA">
7      <requiredservices>
8        <and>
9          <service
10           value_url="http://192.168.0.2/lamp4554"/>
11         <or>
12           <service
13             value_url="http://192.168.0.12/ums"/>
14           <service
15             value_url="http://192.168.0.32/ims"/>
16         </or>
17       </and>
18     </requiredservices>
19   </composite>
20 </compositelist>
```

Listing 5.1: "Simple power saving composite"

5.2 Reliability Approximator

The Reliability Approximator is responsible for approximating the reliability of services in the Homeport system. The Homeport service infrastructure is based on the HTTP protocol [12] and the principles of Representational State Transfer (REST) [13]. One of the guiding principles of REST is that service functionality is only accessible through a fixed set of methods with well-defined semantics. For example, the `GET` method retrieves a representation of a resource and is guaranteed to be safe in the sense that it has to be side-effect free. This property can be used to sample reliability information. Periodically, a `GET` request is issued to all services in the infrastructure, and reliability data is recorded. In the request, the `Cache-Control`-header is used to ensure that the response in not cached. If the service does not respond to the request, it is assumed to be unavailable. To save bandwidth, the sampling frequency can be adjusted and the `HEAD` method can be used instead of `GET`. Once information is collected, it is made available by means of a service. Listing 5.2 shows a listing of sample reliability data.

For some applications, a more subtle notion of reliability is needed. One could, e.g., define a service to be available if it can be turned of, as is the case for the Peak Demand Response scenario. For this purpose, additional reliability information can be added. For example, the `relinfo`-element in line 8 is declared to be a computed

```
<?xml version="1.0" encoding="utf-8"?>
<value timestamp="1268392218.85">
  <service
    value_url="http://192.168.0.12/ums">
    <reliability>
      <relinfo name="HEAD sampled mean" type="sampled" value="0.9"/>
      <relinfo name="HEAD sampled variance" type="sampled" value="0.9"/>
      <relinfo name="turn-off ability" type="computed" value="0.5"/>
      <relinfo name="Product sheet" type="declared" value="0.99"/>
    </reliability>
   </service>
  <service
    ...
   </service>
</value>
```

Listing 5.2: "Reliability data"

reliability value. Finally, a service description can also contain reliability information originating form product specifications. This is also specified by using the `relinfo`-element, here with the `declared`-type.

In retrieving data from the Reliability Approximator, a window can be specified so that only samples obtained within that window is considered. This is necessary to approximate the reliability of services where reliability varies over time.

5.3 Composite Reliability Estimator

Using the reliability data from the Reliability Approximator, the Composite Reliability Estimator can compute reliability data for each composite using the algorithm described in the following.

First time the reliability of a composite s is requested, a function *mk_phi* generates a function phi_s so that, given an environment, *env*, mapping the leaf nodes s_i of s to $E[X_i]$, phi_s calculates the reliability of s. All subsequent requests for reliability data for s, are calculated using phi_s.

The function *mk_phi*, generates phi_s by recursively unfolding s using formula (4.3) and linearizes the resulting expression using formula (4.9). The expression obtained from the linearization process is stored in phi_s. When phi_s is executed with *env*, the expression will be evaluated in the context of *env*. In pseudo-Python, this can be written as shown in listing 5.3.

As previously mentioned, the size of the expression will in worst case be $O(2^n)$, and therefore an alternate version mk_phi_{part} was developed, which use formula (4.12) to only partially linearize the expression. For a composite s having leaf services

```
1 def mk_phi(s):
2   s_lin = linearize(phi(s))
3   phi_s = lambda env: evaluate(env, s_lin)
4   return phi_s
```

Listing 5.3: "The "mk_phi function. phi unfolds an of expression to an expression consisting of sums and products of X's. linearize linearizes the expression into a sum of products. evaluate evaluates an expression using an environment"

$s_1, ..., s_n$, we say that s_i is pure if it only occurs once in s. A node in s is pure if all its leaf services are pure. The `mk_phi_part` function is shown in listing 5.4. The `walk` function recursively traverses s and replaces all pure sub expressions with a function computing the reliability of the sub expression using formula (4.12) (lines 8–14).

To evaluate the worst case performance of *mk_phi* and mk_phi_{part}, we used the functions to generate phi_s for $s = of(s_1, ..., s_n; n/2)$ for varying n, and looked at the space requirements and execution time of phi_s. In figure 5.2, the results of the experiments can be seen. It shows that the execution time for the phi_s generated by *mk_phi* and mk_phi_{part} increase exponentially with n as expected. Note however that s in this experiment is pure for all n, and therefore mk_phi_{part} will not linearize at generation time, but rather generate a phi_s that calculates the value recursively at runtime. Therefore, the space requirements for the phi_s generated by mk_phi_{part} only increase linearly with n, whereas the phi_s generated by *mk_phi* uses exponential space. Considering that *mk_phi* and mk_phi_{part} has comparable performance with respect to time, it can be concluded that mk_phi_{part} is superior to *mk_phi* since is uses significantly less space.

```
1  def mk_phi_part(s):
2    def walk(pures, e):
3      if simple(e):
4        return (s in pures, e)
5      l = [walk(pures, si) for si in e.sub_exps]:
6      if allpure(l): return (True, e)
7      sub_exps = []
8      for (pure, e) in l:
9        if pure:
10         sub_exps.append(
11           lambda env: phi_rec(env, e))
12       else:
13         sub_exps.append(e)
14     e.sub_exps = sub_exps
15     return (False, e)
16
17   pures = pure_services(s)
18   (s_pure, e) = walk(pures, s)
19   if s_pure:
20     phi_s = lambda env: phi_rec(env, e))
21   else:
22     e = linearize(phi(e))
23     phi_s = lambda env: evaluate(env, e)
24   return phi_s
```

Listing 5.4: "The `mk_phi_part` function. `phi_rec` computes the availability recursively using formula (4.12). `pure_services` returns the list of services that only occur once in s"

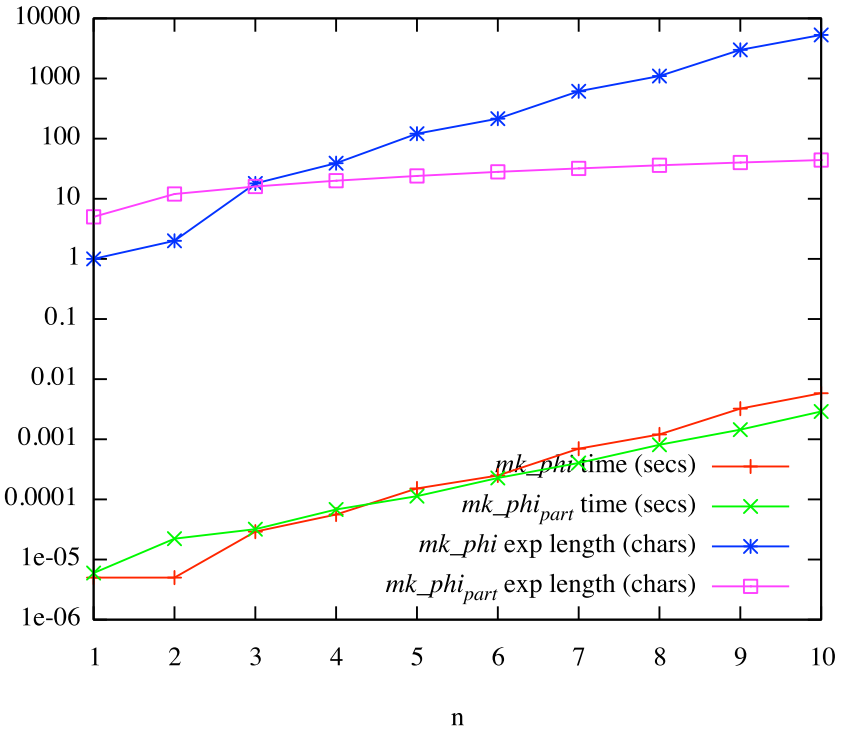

Figure 5.2: Worst case performance of *mk_phi* and mk_phi_{part}

Chapter 6

Evaluation

As mentioned in the introduction, we evaluate our approach in terms of applicability, accuracy, and realizability. We evaluate these in the context of our two scenarios: "peak demand response" (applicability and accuracy) and "saving power" (applicability and realizability).

6.1 Peek Demand Response

To evaluate the accuracy and applicability of our approach, we modelled and analysed data for the peek demand response scenario numerically. Our data comes from the measurement energy usage in a residential home for 23 days from 2010-01-14 to 2010-02-12. For each 15 minutes, we recorded an identity of the device, a time stamp, and the current power consumption of the device. In total this gave 33,595 power measurements and we measured the energy usage of 19 devices.

In the following, we assume that when a device is consuming power, it is relevant for the energy company to turn it off. This means that it is the number of devices consuming power at a given time that determines if a house is available for the energy company (to use in peek demand response handling).

For the first part of the scenario evaluation, we considered the availability of a house service, h, over the 23 days based on our measured data. We calculated and analyzed the following

- *Actual average availability* per day based on measured data, dividing the number of times at least d_{min} devices were available by the total number of measurement times for the day
- *Model average availability* per day based on the measured availability of devices and values of d_{min}

The structure function of the composition is

$$\phi(h) = \phi(of(d_1, ..., d_{19}); d_{min})$$

Since each service only appears once in the composition, e, we applied equation (4.12) to obtain the model availability.

In the context of resource-constrained devices, the frequency with which we monitor service availability (e.g., using HTTP HEAD requests as described in the Implementation section) is important. We analyzed this by varying the subset of power measurements we used with a sampling rate parameter, δ. For $\delta = 2$, e.g., we used every second power measurement in calculating model availability. A plot of a subset of these calculations is shown in Figure 6.1.

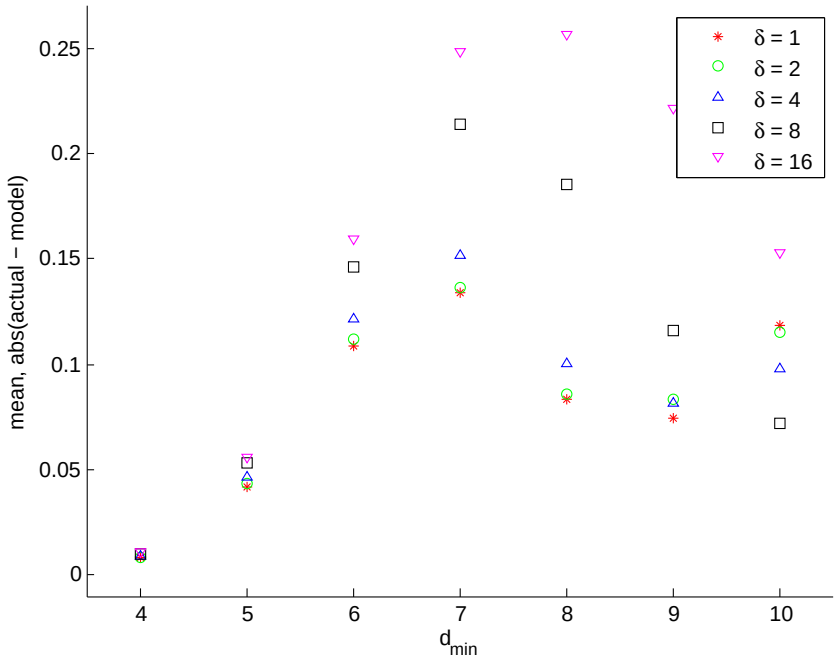

Figure 6.1: Accuracy of model availability for house service, h

The figure plots sampling rate, δ, against the mean accuracy of model availability, i.e., the mean of the absolute difference between actual availability and model availability. For a sampling rate $\delta \leq 4$, the accuracy is within 0.15, but for $\delta > 4$ the accuracy less than 0.20. We thus conclude that for small sampling intervals in the scenario, the model availability is reasonable.

Our next step in the evaluation was to interpret the data differently and in a slightly artificial way to test the scalability of our approach: each day of measurements were interpreted as data from one of 23 different houses and the data used in the full

scenario. Here the model of e gives a structure function:

$$\phi(e) = \phi(of(h_1, ..., h_{23}; h_{min}))$$

with

$$\phi(h_i) = \phi(of(d_{i,1}, ..., d_{i,19}; d_{min}))$$

As in the previous step, we may calculate model availability by applying (4.12). We assumed a sample rate of $\delta = 4$ (cf. the first part of the evaluation) and varied d_{min} and h_{min} and compared to actual availability. Part of the results are plotted in Figure 6.2.

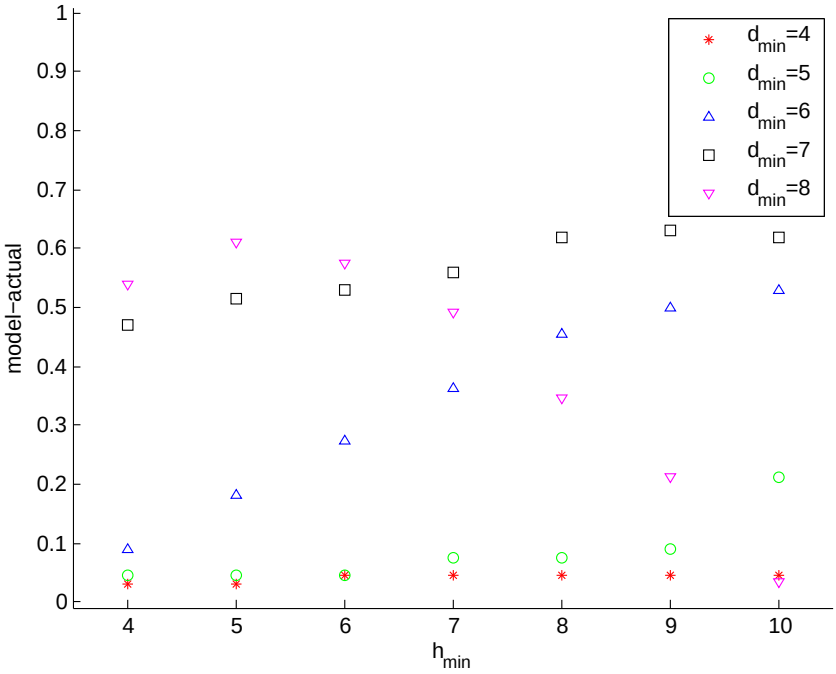

Figure 6.2: Accuracy of model availability for complete peak demand response service, e

In the figure, the accuracy of model availability for e is plotted against h_{min} for selected values of d_{min}. For simple structure functions ($d_{min} \leq 5$ and $h_{min} \leq 9$), we see that in the scenario, our model gives a reasonable accuracy of model availability (< 0.10). In conclusion, we may say that our model provides reasonable accuracy of predictions for simple models with a reduced availability sampling rate (in the scenario of $\delta = 4$). However, further research and analysis is needed in order to translate this into concrete guidelines on when model availability is sufficient.

6.2 Saving Power

The basic version of the Saving Power scenario was implemented as a composite service running on the NSLU2 device interacting with emulated light and motion sensor services. The Reliability Approximator sampled the services while we injected failures. The composite was described using CSDF and the Composite Reliability Estimator used the data from the Reliability Approximator to estimate the reliability of the composite service over a time period. Implementing the Saving Power scenario demonstrates that our approach to reliability estimation is applicable and realizable.

It was observed in the previous section that the mk_phi_{part} function uses linear space if all services in the composite are pure. To further evaluate the space requirements of the mk_phi_{part} function, we considered two variants of the basic Saving Power scenario that might be more realistic than the composite service used in the previous section.

The first variant is an aggregation of a set of basic composites. Let $sps_n =$

$$
\begin{aligned}
&of(\\
&\quad of(s_1, of(s_2, s_3; 1); 2),\\
&\quad ...,\\
&\quad of(s_{3n-2}, of(s_{3n-1}, s_{3n}; 1); 2)\\
&\qquad\qquad\qquad\qquad\qquad\qquad ; n)
\end{aligned}
$$

sps_n is still pure, and in general this will not be the case in a home, because services will be used in multiple composites. Therefore, we let $sps'_n =$

$$
\begin{aligned}
&of(\\
&\quad of(s_1, of(s_2, s_4; 1); 2),\\
&\quad of(s_3, of(s_4, s_6; 1); 2),\\
&\quad ...,\\
&\quad of(s_{2n-1}, of(s_{2n}, s_{2n+1}; 1); 2)\\
&\qquad\qquad\qquad\qquad\qquad\qquad ; n)
\end{aligned}
$$

Again, sps'_n is an aggregate of a set of basic composites, but now each term shares a service with the next term, so no term is pure. This could model the situation that each motion sensor detects motion in two rooms.

With varying values of n we generated phi_s using mk_phi_{part} and estimated the amount of space required. The result can be seen in figure 6.3. Note that n here is the number of terms inside the outermost of. It can be observed that while the amount of required space for phi_{sps_n} grows linearly with n, the amount of space required for $phi_{sps'_n}$ grows exponentially with n.

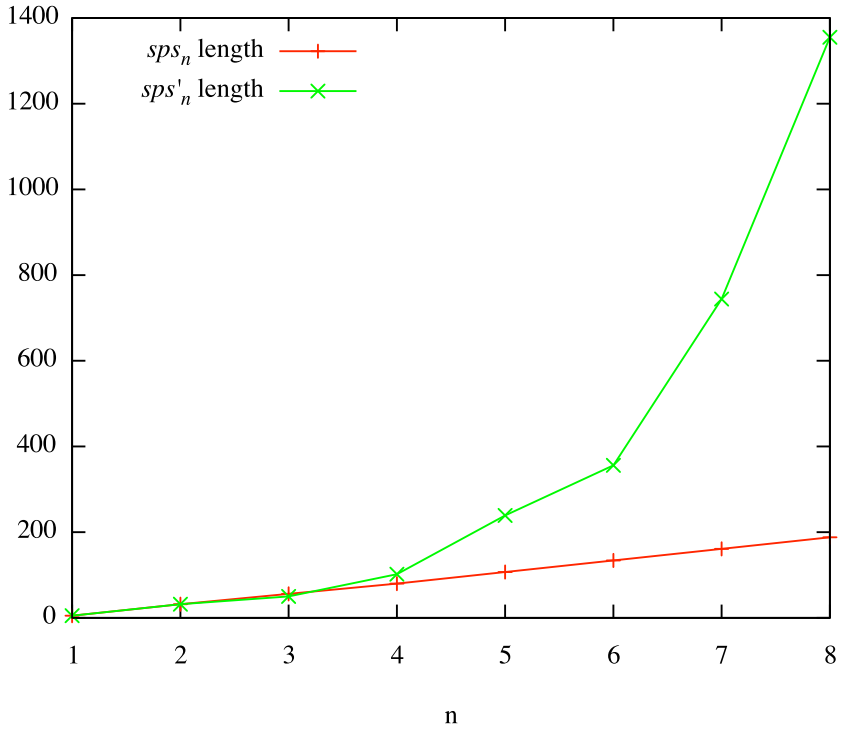

Figure 6.3: Space requirements for phi_s, $s = sps_n$ and $s = sps'_n$

Chapter 7

Future Work

As discussed in the modelling and implementation sections, both or algorithm and implementation have exponential worst case performance. Many service compositions will primarily be based on "or" and "and" composition in which case the space and time requirements are only linear in the number of services in the composition. However, there are still issues of scalability in certain composition, in particular if a service is part of several sub-compositions. Consider a house example again, a light sensor service, s, could be part of two compositions c_1 and c_2 for which the structure functions are thus not independent. One area of future work would then be to investigate techniques for modelling dependent failures and in particular the β-factor model [14] could be interesting since the sensor s is a source of common cause failures in c_1 and c_2.

Further in this direction, it is also interesting to investigate other statistical properties of reliability than (mean) availability. The variance of Mean Time To Failure/Mean Time To Repair (MTTF) could be captured and together with their distribution be used to calculate confidence intervals for our estimates. Furthermore, it could be interesting to use context-awareness also in the estimation of reliability; wireless technologies, e.g., typically behave differently according to ambient temperature [15]. More complex reliability models, e.g., based on Markov chains [26] is also a possible further direction although that would go somewhat against our realizability challenge in that reliability estimation would become more complex.

We have implemented our approach in the HomePort service framework, but the principles of the model can be realized in other frameworks. A part of this would be to generalize our concrete (XML-based) description so that it can extend existing composition descriptions. An example of this would be describing the reliability requirements of services composed using the Service Component Runtime (i.e., the Declarative Services specification) of OSGi [19]. In particular, the distributed R-OSGi framework in which resource-constrained devices may be used as OSGi services would be interesting to use in the context of pervasive computing [27].

Finally, our approach needs further evaluation in real deployments. We are currently working on enabling the use of HomePort on ZigBee devices which could be the basis of an evaluation. Even though it is not strictly necessary to run HomePort software on the actual devices, it will make it easier to, e.g., inject faults into running services.

Chapter 8

Conclusion

Service composition is a key element in realizing pervasive computing applications. By aggregating multiple device functionalities, novel and innovative uses of IT can be realized. With the power of distributed resources comes also the weakness of multiple points of failure, and therefore it is important to be able to reason about dependability of service compositions. In this report, we presented an approach for estimating reliability of composite services. The approach consists of a theoretical model and a practical implementation that, given a description of the dependability structure of a service composite, can automatically compute an estimated reliability based on previously sampled measurements of the services constituting the composite. We demonstrated that the approach is applicable by using it on two relevant and realistic scenarios. The approach was shown to be reasonable accurate by comparing estimated values with actual measurements, and finally, by implementing the approach in a service infrastructure for home automation, we demonstrated that it is realizable.

Acknowledgements

The work described in this report has been partially funded by the "Dit Hus" project[1]. We thank zensehome ApS [2] for the data used in our evaluation and Mads Ingstrup for comments on an earlier version of this report.

[1] http://www.cs.au.dk/dithus/

[2] http://www.zensehome.dk/

Bibliography

[1] S.L. Amundsen and F. Eliassen. A resource and context model for mobile middleware. *Personal and Ubiquitous Computing*, 12(2):143–153, 2008.

[2] A. Avižienis, J.C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, pages 11–33, 2004.

[3] J.E. Bardram and N. Nørskov. A context-aware patient safety system for the operating room. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 272–281. ACM, 2008.

[4] Luciano Baresi, Elisabetta Di Nitto, Carlo Ghezzi, and Sam Guinea. A framework for the deployment of adaptable web service compositions. *Service Oriented Computing and Applications*, 1(1):75–91, 2007.

[5] J. Brandt and S.R. Klemmer. Lash-Ups: A Toolkit for Location-Aware Mash-Ups. In *Proceedings of the 19th annual ACM Symposium on User Interface Software and Technology (Montreux, Switzerland, October 15–18, 2006). UIST*. Citeseer, 2006.

[6] Jeppe Brønsted, Klaus Marius Hansen, and Mads Ingstrup. Service composition issues in pervasive computing. *IEEE Pervasive Computing*, 9(1):62–70, 2010.

[7] Jeppe Brønsted, Per Printz Madsen, Arne Skou, and Rune Torbensen. The HomePort System. In *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, jan. 2010.

[8] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):281–308, 2004.

[9] Transaction service, version 1.4. `http://www.omg.org/technology/documents/formal/transaction_service.htm`, 2003. Accessed 2010-03-01.

[10] A. Dan and P. Narasimhan. Dependable Service-Oriented Computing. *IEEE Internet Computing*, 13(2):11–15, 2009.

[11] W.K. Edwards and R.E. Grinter. At home with ubiquitous computing: Seven challenges. *Lecture Notes In Computer Science*, pages 256–272, 2001.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817.

[13] Roy T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[14] KN Fleming. A reliability model for common mode failures in redundant safety systems. Technical report, GA-A–13284, General Atomic Co., San Diego, Calif.(USA), 1974.

[15] W.C. Hua, H.H. Lai, P.T. Lin, C.W. Liu, T.Y. Yang, and G.K. Ma. High-linearity and temperature-insensitive 2.4 GHz SiGe power amplifier with dynamic-bias control. In *IEEE RFIC Symp. Dig*, pages 609–612, 2005.

[16] Valerie Issarny, Daniele Sacchetti, Ferda Tartanoglu, Francoise Sailhan, Rafik Chibout, Nicole Levy, and Angel Talamona. Developing ambient intelligence systems: A solution based on web services. *Automated Software Engineering*, 12(1):101+, 2004.

[17] A. Jhingran. Enterprise information mashups: integrating information, simply. In *Proceedings of the 32nd international conference on Very large data bases*, page 4. VLDB Endowment, 2006.

[18] Swaroop Kalasapur, Mohan Kumar, and Behrooz A. Shirazi. Dynamic service composition in pervasive computing. *Parallel and Distributed Systems, IEEE Transactions on*, 18(7):907–918, 2007.

[19] Peter Kriens, editor. *OSGi Service Platform, Service Compendium, Release 4, Version 4.2*. aQute Publishing, 2009.

[20] JCC Laprie, A. Avizienis, and H. Kopetz. *Dependability: Basic concepts and terminology*. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1992.

[21] N.G. Leveson. *Safeware: system safety and computers*. ACM New York, NY, USA, 1995.

[22] X. Liu, Y. Hui, W. Sun, and H. Liang. Towards service composition based on mashup. In *IEEE Congress on Services*, pages 332–339, 2007.

[23] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, et al. Bringing semantics to web services: The OWL-S approach. *Lecture Notes in Computer Science*, 3387:26–42, 2005.

[24] Nikola Milanovic and Miroslaw Malek. Current solutions for web service composition. *IEEE Internet Computing*, 8(6):51–59, 2004.

[25] S.B. Mokhtar, J. Liu, N. Georgantas, and V. Issarny. QoS-aware dynamic service composition in ambient intelligence environments. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 317–320, 2005.

[26] Marvin Rausand and Arnljot Høyland. *System Reliability Theory: Models and Statistical Methods*. New York: John Wiley & Sons, 2nd edition, 2004.

[27] Jan S. Rellermeyer, Michael Duller, Ken Gilmer, Damianos Maragkos, Dimitrios Papageorgiou, and Gustavo Alonso. The software fabric for the internet of things. In Christian Floerkemeier, Marc Langheinrich, Elgar Fleisch, Friedemann Mattern, and Sanjay E. Sarma, editors, *The Internet of Things, First International Conference, IOT 2008, Zurich, Switzerland, March 26-28, 2008. Proceedings*, volume 4952 of *Lecture Notes in Computer Science*, pages 87–104. Springer, 2008.

[28] AM Rushdi. Utilization of symmetric switching functions in the computation of k-out-of-n system reliability. *Microelectronics and Reliability*, 26(5):973–987, 1986.

[29] AM Rushdi. Comments on: 'An efficient nonrecursive algorithm for computing the reliability of k-out-of-n systems' by AK Sarje and E. V. Prasad. *IEEE Transactions on Reliability*, 40(1):60–61, 1991.

[30] J. P. Sousa, V. Poladian, D. Garlan, B. Schmerl, and M. Shaw. Task-based adaptation for ubiquitous computing. *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, 36(3):328–340, 2006.

[31] Yuping Yang, Fiona Mahon, M. Howard Williams, and Tom Pfeifer. Context-aware dynamic personalised service re-composition in a pervasive service environment. In *Proceedings of Ubiquitous Intelligence and Computing*, volume 4159 of *LNCS*, pages 724–735. Springer, 2006.

[32] Apostolos Zarras, Panos Vassiliadis, and Valérie Issarny. Model-driven dependability analysis of webservices. In Robert Meersman and Zahir Tari, editors, *CoopIS/DOA/ODBASE (2)*, volume 3291 of *Lecture Notes in Computer Science*, pages 1608–1625. Springer, 2004.

Appendix A

CSDF XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <xs:element name="compositelist">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="composite"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="composite">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="requiredservices"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:NCName"/>
      <xs:attribute name="language" use="required" type="xs:NCName"/>
      <xs:attribute name="name" use="required" type="xs:NCName"/>
      <xs:attribute name="src" use="required" type="xs:anyURI"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="requiredservices">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="1">
        <xs:element ref="service"/>
        <xs:element ref="and"/>
        <xs:element ref="or"/>
        <xs:element ref="kofn"/>
      </xs:choice>
    </xs:complexType>
```

```
34  </xs:element>
35
36  <xs:element name="service">
37    <xs:complexType>
38      <xs:attribute name="id" use="required" type="xs:NCName"/>
39      <xs:attribute name="name" use="required" type="xs:NCName"/>
40      <xs:attribute name="type" use="required" type="xs:NCName"/>
41      <xs:attribute name="unit" use="required" type="xs:NCName"/>
42      <xs:attribute name="value_url" use="required" type="xs:anyURI"/>
43    </xs:complexType>
44  </xs:element>
45
46  <xs:complexType name="requirementType">
47    <xs:choice minOccurs="1" maxOccurs="unbounded">
48      <xs:element ref="service"/>
49      <xs:element ref="and"/>
50      <xs:element ref="or"/>
51      <xs:element ref="kofn"/>
52    </xs:choice>
53    <xs:attribute name="required" type="requiredType"/>
54  </xs:complexType>
55
56  <xs:simpleType name="allType">
57    <xs:restriction base="xs:string">
58       <xs:enumeration value="all"></xs:enumeration>
59    </xs:restriction>
60  </xs:simpleType>
61
62  <xs:simpleType name="requiredType">
63    <xs:union memberTypes="xs:positiveInteger allType"/>
64  </xs:simpleType>
65
66  <xs:element name="and" >
67    <xs:complexType>
68      <xs:complexContent>
69        <xs:restriction base="requirementType">
70          <xs:choice minOccurs="1" maxOccurs="unbounded">
71            <xs:element ref="service"/>
72            <xs:element ref="and"/>
73            <xs:element ref="or"/>
74            <xs:element ref="kofn"/>
75          </xs:choice>
76          <xs:attribute name="required" type="requiredType" fixed="all"/>
77        </xs:restriction>
78      </xs:complexContent>
79    </xs:complexType>
80  </xs:element>
81
82  <xs:element name="or" >
```

```
83    <xs:complexType>
84     <xs:complexContent>
85      <xs:restriction base="requirementType">
86       <xs:choice minOccurs="1" maxOccurs="unbounded">
87        <xs:element ref="service"/>
88        <xs:element ref="and"/>
89        <xs:element ref="or"/>
90        <xs:element ref="kofn"/>
91       </xs:choice>
92       <xs:attribute name="required" type="requiredType" fixed="1" />
93      </xs:restriction>
94     </xs:complexContent>
95    </xs:complexType>
96   </xs:element>
97
98   <xs:element name="kofn" >
99    <xs:complexType>
100    <xs:complexContent>
101     <xs:restriction base="requirementType">
102      <xs:choice minOccurs="1" maxOccurs="unbounded">
103       <xs:element ref="service"/>
104       <xs:element ref="and"/>
105       <xs:element ref="or"/>
106       <xs:element ref="kofn"/>
107      </xs:choice>
108      <xs:attribute name="required" use="required" type="requiredType"/>
109     </xs:restriction>
110    </xs:complexContent>
111   </xs:complexType>
112  </xs:element>
113
114
115 </xs:schema>
```

Listing A.1: "CSDF XML schema"