Declarative Specification of Service Management Attributes

Vitalian A. Danciu, Nils gentschen Felde, Martin Sailer
Munich Network Management Team
University of Munich
Oettingenstrasse 67, D-80538 Munich, Germany
Email: {danciu|felde|sailer}@mnm−team.org

***Abstract*— Providers, operators and customers understand that the concept of an IT service offers a beneficial abstraction from actual IT operations, effectively encapsulating the provisioning of the service. Yet, IT services are in fact provided by installations composed of very large numbers of managed elements. Hence, management of a service implies the management of the multitude of elements and sub-services upon which the service relies. The mapping of a service's attributes onto resources to "make the service visible" is as much of a challenge as the execution of service management actions (i.e. actions directed at a service) on the underlying infrastructure. Even today, practical solutions to these issues are scarce.**

In this paper, we propose a methodology for the synthesis of service attributes to create the foundation for a service management information base. We present a declarative language suitable for the representation of service attributes in dependence of management attributes of the provisioning infrastructure. In addition, we discuss a service monitoring architecture driven by service attribute specifications.

I. Introduction

In the last decades, effective concepts have been developed and deployed to cope with the management of elements and systems. These concepts were facilitated by the simple base structure of elements and systems—from a management point of view—and by the idea of the managed object (MO). They constitute the foundation of current management systems by providing a common representation of the devices or systems to be placed under IT management [11]. Such tools have enabled an increasingly efficient control of resources despite their increasing complexity and the growing number of devices.

It is desirable to adapt the same concepts for the use with services. In the same way the managed objects representing resources are specified in a management information base (MIB), services could be described by creating a *Service Management Information Base* (SMIB). However, service management suffers from the complexity inherent to services.

The presence of a suitable view on a target of management is prerequisite to effective management of that target. However, unlike resources and, to some extent systems and networks, services are less palpable entities. They can be described as the result of the operation of a compound of resources including devices, applications and persons. Most of the technical information needed to describe a service is already available with the resources that compose the service. As resource monitoring is in place pervasively, this information is available at a technical level, in a manner reusable by service management.

We illustrate our requirements to service description by means of a simplified management scenario taken from the Grid community in Section II. Grids are well-known for their complex, service-based structure and pose additional challenges (such as inter-domain service provisioning) compared to the IT services commonly encountered.

A service can be described by a set of attributes—in analogy to the attributes defined in resources' MIBs. Attribute definitions for a number of general purpose attributes have been specified in the literature. In most cases, however, it is mandatory to assemble service-proprietary attributes in order to describe a service. A generic approach to attribute definition for managed services is still missing. An inherent challenge when attempting to specify service management attributes is the virtually infinite number of possibilities when constructing a service. Even the "standard" internet services can be provisioned in a vast variety of different ways. We discuss the structure of service attributes in Section III and propose a methodology for the synthesis of service attributes.

Taking into account the associations between the resources used in service provisioning is paramount when attempting to describe services in a formal manner. Based on the concepts presented in Section III, we have developed the Service Information Specification Language (SISL), a declarative language suited to express service attributes in dependence of management data gathered from resources. We describe SISL in detail in Section IV.

In any IT management setting, the amount of change to the deployed infrastructure made necessary by an approach is an important benchmark for the approach itself. In Section V, we discuss an architecture aiming to leverage the deployed base of management tools while providing a realisation of the service view.

Service management is not a new discipline and our approach has drawn on existing concepts. In Section VI, we give an overview of related work with regard to information modelling as well as related formal languages and approaches. We conclude the paper in Section VII with a discussion of the open issues remaining.

II. Scenario and Requirements

Due to their highly distributed nature, the number of different stakeholders from different administrative and legislative domains, Grids exhibit all but overwhelming management needs. Services are provided by autonomous entities thus barring inter-domain management access at a more technical level.

A. A broader view

The D-Grid project (http://www.d-grid.de) infrastructure is funded by the German Federal Ministry of Education and Research (BMBF) and reaches for providing a robust, flexible and sustainable Grid infrastructure for scientific purposes. This Grid is used by so-called communities, one among them the HEP (High Energy Physics) Community Grid performing computations using the tremendous amount of data produced by the Large Hadron Collider (LHC) at CERN in Switzerland. It is expected to produce about 15 Petabytes of data each year which needs to be stored and analysed by thousands of scientists working for different organisations at different locations. All of the participating organisations provide storage and computing resources in order to store the data produced at CERN redundantly and to provide enough computing power for the analysis by the researchers around the world.

B. Scenario details

The simplified scenario is adapted from the computing service provided within Grids. It motivates the service management requirements that determine the goals of our work.

A Grid computing service is composed of several computing services provided by different sites. Thus, the status of the Grid computing service depends on the statuses of the computing services it relies on. In turn, these computing services depend on the components required for their operation and thus their status depends on the underlying infrastructure and components. In our simplified example sketched in Figure 1 a local computing service depends on a router, the DNS service and the local nodes providing the actual computing power.

From the perspective of a Grid user, the "Grid itself" provides a computing service. In contrast, the providers of the Grid computing service rely on services hosted at the different Grid sites and technically usually provided to them by an agreed upon Grid middleware.

C. Management Challenges and Requirements

Aided by current management systems, we are able to acquire information about the managed objects within the management domain. This information represents the isolated state of the resources being managed. In the case of our scenario (Figure 1), we can for example easily determine the status of the DNS, router or any of the computing nodes by means of a management tool or direct manual access. However, we are interested in the status of the computing *service* provided by means of these resources—not in the status of each single resource.

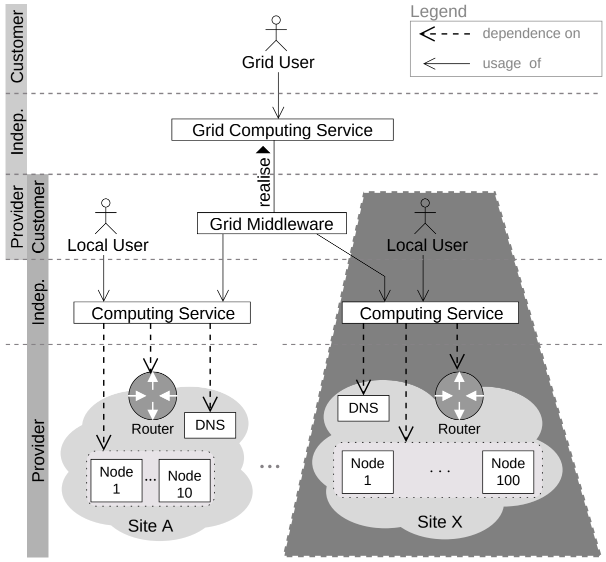

Fig. 1. A simple Grid computing scenario

The provisioning of Grid services is distributed between different autonomous domains that employ different technologies, management techniques and specify different policies for access to and use of resources. Therefore, any approach targeted at the Grid community must be generic in terms of technology and light on requirements imposed onto sites wishing to implement it.

An approach suitable for our scenario should therefore

- address the structure of attributes describing a service,
- support definition of attributes for any given service,
- support translation into a unified format for the data gathered from resources,
- allow reuse of existing resource management tools,
- lead to the production of service attribute values that represent the state of a service at a certain point in time,
- be technology independent to the highest possible degree.

D. Scope

Structurally, the scenario setup follows the MNM Service Model [8] as sketched in Figure 2. The applicable roles according to the service model are indicated by the bars on the left-hand side in Figure 1.

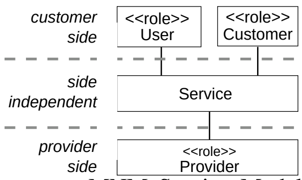

Fig. 2. MNM Service Model (Bird's eye view)

Note that the bars overlap, indicating different roles for one entity depending on the context (local or Grid) of the service. We apply the service model roles to decompose the Grid service. Most of this paper focuses on one occurrence of a service, as denoted by the darker background trapeze; for a discussion regarding the nested service (i.e. the Grid service as a whole) the reader is referred to Section VII-.4.

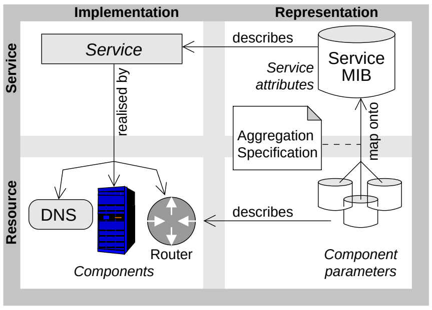

Fig. 3. Interrelationships of services and components

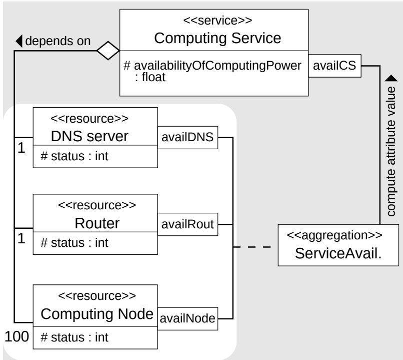

Fig. 4. Example model of service attribute dependencies

III. Synthesising service attributes

A key to meeting the requirements delineated in the previous section is a comprehensive, management-oriented description of services. In this section, the concepts required to facilitate such a description are presented. Based on these concepts, a methodology for the generic specification of service attributes is proposed.

A. Attribute-based service definition

Services represent an abstraction of a collection of resources that can be treated as a single entity for management tasks. They can, in effect, be managed, reported on and visualised in a similar manner to physical resources. This allows a clearer association between services and customers to be made and thus a better alignment to customers' requirements.

Figure 3 illustrates in principle a management setup for a service. The service draws on components described by component parameters. The latter are traditionally provided by management agents implementing a component MIB. The items in the top right region of the figure are part of the approach presented in this paper. The goal is to provide access to a service's description in the same manner as is common in network and systems management.

The Service Management Information Base (SMIB) that contains a description of the service is supported by the component parameter values of the components providing (parts of) the service. Any aspect of the service may be dependent on several component parameters. We call such an aspect of a service a *service attribute*. The aggregation instruction associated with the path between component parameters and SMIB specifies the manner in which those parameters are aggregated to form a service attribute. Note that the internet management approach is a possible but not mandatory reification of the SMIB idea. We intend the SMIB to be a conceptual container for service-related management information.

1) Example: A model of the example service presented in Section II shows the dependencies between components providing the `ComputingService` as well as the qualified attributes relevant to the aggregation (Figure 4). In this case, the availability of the service `availCS` is dependent on the combined availabilities of DNS `availDNS`, router `availRout` and computing nodes `availNode`. The resulting attribute value is calculated by means of the *ServiceAvail* aggregation.

TABLE I
ATTRIBUTE DEFINITION

Name	totalCurrentlyAvailableComputingPower
Description	The computing power available to service users.
Synonym	N/A
Type	float
Unit	*none*
Constraint	$0 \leq$ computingPowerAvailable ≤ 1
Related Component Parameters	Router (Availability $\rightarrow R$) DNS (Availability $\rightarrow D$) Node* (Availability, Strength $\rightarrow a_i, p_i$)
Aggregation	$D \cdot R \cdot \frac{\sum_{i=1}^{N}(p_i a_i)}{\sum_{i=1}^{N} p_i}$

2) Characteristics of a service attribute: In the general case, a service attribute has the properties enumerated in the leftmost column of Table I. While the *Name* field of the attribute can be used for its identification by both human and machine actors, the *Description* and *Synonym* fields are designed to transport semantics in a human-readable form. The *Description* field contains a textual description of the purpose of the attribute while the *Synonym* field stores the names of equivalent attribute definitions from information models other than the one employed. It is intended to facilitate transition between information models or the concurrent use of several such models. The *Type* and *Unit* fields refer to the value of the service attribute and the *Constraint* expression may specify a valid range.

The *Aggregation* rule, often expressed as the right hand side of an equation, specifies how the service attribute value is to be computed from the component attribute values that it depends on. The components involved in the aggregation clause are to be specified in the *Related Component Parameters* field.

A formal specification of service attributes can be realised by means of modern information modelling frameworks (e.g. those discussed in section VI). In most cases, this requires augmentation of the employed modelling framework by adding the ability to express aggregation instructions and constraints.

B. Methodology for service attribute synthesis

To synthesise attribute definitions for a service we have devised a methodology dividable into four phases (see Figure 5). We describe each of these phases in short and focus on the *define* phase that is shown in detail in the figure.

1) Derive: The primary aim of the first phase is to determine the information requirements for characterising a service. Such requirements can be derived by analysing customers' requirements and Service Level Agreements (SLAs) as well as management frameworks, e.g. OSI's FCAPS and ITIL (see Section VII).
Example: An attribute relevant to the service described in our scenario is the availability of the Computing Service. In this case, the attribute is not actually derived systematically. In many typical management cases however, the service attributes result from the management requirements of the IT organisation.

2) Define: Based on the information requirements identified in the previous phase, service attributes are defined using a formal language. Such definitions can be realised by means of information modelling frameworks (e.g. CIM [6], SID [21], SMI [16]). In addition to a static description of the service at hand, this includes aggregation instructions as well as measurement parameters. Accordingly, the define phase can be broken down into the following sub-steps:

1) *Declaration of invariant fields.* Static data about a service attribute (e.g. name/id and a textual description) is determined and the base structure of the attribute is created.
   Example: The name of the attribute (`total-CurrentlyAvailableComputingPower`) as well as its textual description (see Table I) can be determined a priori. That also holds true for the attribute's type (in this case floating point) and unit.
2) *Assessment of service component dependencies.* The composition of a service is either explicitly documented in a model or management knowledge of the service administrators. When employing an information modelling framework, (functional) dependencies are represented as associations. Identifying these associations therefore yields the list of candidate components supporting the service.
   Example: Considering the dependencies in the model in Figure 4, we can establish that the service is dependent on the DNS service, the router and the computing nodes.
3) *Identification of relevant component parameters.* Some of the parameters used to describe the components identified in the previous step may influence the service attribute at hand. Identifying these parameters therefore constitutes a required step to the definition of aggregation instructions.
   Example: Our computing service cannot be accessed if the router or DNS server are unavailable. Also, obviously, the presence of available computing nodes impacts the availability of the service as a whole.
4) *Declaration of measurement parameters.* In order to support monitoring of the component parameters identified in the former step, a number of instructions on how actual measurement should be carried out is specified. This includes parameters such as sampling rate and number of samples to be acquired as well as the data format, API and protocol.
   Example: In our case, it is merely required to test the state (up or down) of the components identified in the previous step. The method employed to test their state is highly dependent on the available management tools. The resource declarations in Figure 10 show one possible declaration applicable to our scenario.
5) *Specification of aggregation rules.* After having identified the relevant component parameters, it is declared how these parameters are to be combined in order to form the service attribute in question.
   Example: For our service to be available, router and DNS service must be operational. In addition, the number of live computing nodes and their computational power determine the amount of computing power available to users. The formula in Table I reflects an applicable aggregation rule.

3) Monitor: To compose an integrated view of the service attribute's state, the (component-oriented) management data gathered by network and systems management tools (e.g. Nagios, Cacti, etc.) needs to be combined. This can be addressed by an architecture capable of aggregating data produced by existing tools according to the aggregation and measurement instructions given in the define phase of a service attribute (see Section V).

4) Use: Service attribute values provided by the service monitoring architecture can be leveraged in management applications. To describe a complete service, the attributes are compiled into groups corresponding to that service. These groups constitute the contents of the Service MIB.

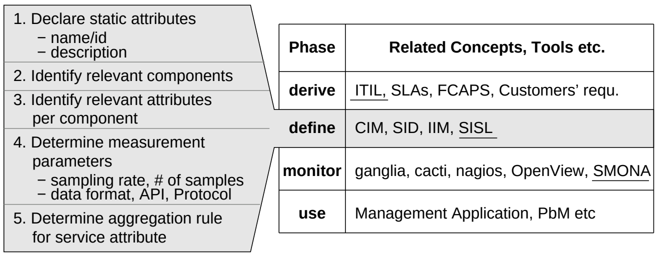

Fig. 5. Methodology overview

IV. SISL: A LANGUAGE FOR SERVICE INFORMATION SYNTHESIS

The *Service Information Specification Language* (SISL) is a formal language capable of expressing in a declarative manner

1) resource attributes relevant to a service,

<table>
<tr><td>⟨aggregation⟩::=</td><td>”aggregation{” ⟨identifier⟩ ⟨description⟩ ⟨resource⟩ ⟨function⟩ ⟨notification⟩ ”}”</td></tr>
<tr><td>⟨description⟩::=</td><td>”description{” [⟨author⟩] [⟨date⟩] [⟨text⟩] ”}”</td></tr>
<tr><td>⟨identifier⟩::=</td><td>”id = ” ⟨id⟩</td></tr>
<tr><td>⟨id⟩::=</td><td>⟨string⟩</td></tr>
<tr><td>⟨author⟩::=</td><td>”author{” ⟨string⟩ ”}”</td></tr>
<tr><td>⟨date⟩::=</td><td>”date{” ⟨dateString⟩ ”}”</td></tr>
<tr><td>⟨text⟩::=</td><td>”text{” ⟨string⟩ ”}”</td></tr>
</table>

Fig. 6. SISL Basic Expressions

2) data sources providing such resource attributes,
3) aggregation operations to be performed to synthesise a service attribute from resource data, as well as
4) constraints to be applied to the acquisition of data.

It is designed to be generic in regard to both services and technologies in order to allow its use with every service, dependent on any resource or sub-service. To allow the exploitation of existing, deployed management tools, the use of such tools as data sources has been considered in the language design. SISL is XML-based to facilitate its integration into management tools.

In the following, we will describe the most important syntactic structures of the language and explain their objective and use. In order to avoid using hard-to-read XML listings, we will give the SISL syntax in a simplified notation that is equivalent to its XML grammar. The expressions in braces correspond to the content of an XML element, while the token preceding the opening brace corresponds to the name of that element. Attributes are given using a “name=value” notation at the beginning of an element’s contents.

A. Basic expressions

The base element of SISL is the `aggregation` that can be employed to represent a service attribute. An aggregation encapsulates specifications regarding data sources, processing instructions for the data gathered as well as conditions pertaining to the delivery of data. Figure 6 shows the basic structure of an SISL aggregation in Extended Backus Naur Form (EBNF). An aggregation is composed of a declarations of `resource(s)` (see IV-B), `function(s)` (see IV-C), a `notification` (see IV-D) and a `description`.

SISL is a typed language supporting integers, floating point numbers, strings, temporal expressions (date, time) as well as boolean. The precision of the types (and thus the value range of a type) is intentionally left unspecified as it would either require a large number of different types (e.g. short and long integers) or force assumptions regarding the range of the values originating with a data source.

B. Resource and data related expressions

SISL design assumes that data have to be acquired from the underlying resources periodically. Therefore, the resources which shall be polled have to be declared and the attributes which are of further interest have to be specified. An

<table>
<tr><td>⟨resource⟩::=</td><td>[⟨resource⟩] ”resource{” ⟨identifier⟩ [⟨description⟩] ⟨source⟩ ⟨sourceAttrib⟩ ”}”</td></tr>
<tr><td>⟨source⟩::=</td><td>”source{” ⟨string⟩ ”}”</td></tr>
<tr><td>⟨sourceAttrib⟩::=</td><td>[⟨sourceAttrib⟩] ”sourceAttrib{” ⟨id⟩ ⟨interval⟩ ⟨return⟩ ”}”</td></tr>
<tr><td>⟨interval⟩::=</td><td>”interval{” ⟨float⟩ ”}”</td></tr>
<tr><td>⟨return⟩::=</td><td>”return{” ⟨integer⟩ | ⟨float⟩ | ⟨boolean⟩ | ⟨date⟩ | ⟨string⟩ ”}”</td></tr>
</table>

Fig. 7. SISL Resource and Data Related Expressions

<table>
<tr><td>⟨function⟩::=</td><td>”function{” ⟨identifier⟩ ⟨description⟩ ⟨method⟩ ⟨parameters⟩ ⟨return⟩ ”}”</td></tr>
<tr><td>⟨method⟩::=</td><td>”method{” ”sum” | ”diff” | ”mult” | ”div” | ... ”}”</td></tr>
<tr><td>⟨parameters⟩::=</td><td>”parameters{” ⟨valueset⟩ ”}”</td></tr>
<tr><td>⟨valueset⟩::=</td><td>[⟨valueset⟩] ”valueset{” ⟨resourceRef⟩ | ⟨functionRef⟩ ”}”</td></tr>
<tr><td>⟨resourceRef⟩::=</td><td>”resourceRef{’” ⟨id⟩ ”@” ⟨id⟩ ”’,” ⟨integer⟩ ”}”</td></tr>
<tr><td>⟨functionRef⟩::=</td><td>”functionRef{’” ⟨id⟩ ”’}”</td></tr>
</table>

Fig. 8. SISL Processing Instructions

`aggregation` may include one or more resource entries and one resource may deliver one or more attribute value pairs.

Figure 7 shows the grammar of a resource entry. Mainly, the `source` specifies the resource which serves as information providing entity and the `sourceAttrib` defines the attributes necessary for further aggregation. The `interval` specifies the period of time between queries to the resource.

C. Processing instructions

To make a statement about a service aspect, the data acquired from resources has to be processed. For example, the actual value of a resource attribute may be less relevant to a service compared to the variance of the value. In many cases, sampled values need to be consolidated into medians, or sums be computed over a number of values. SISL `functions` (Figure 8) address this requirement. They consist of a set of `parameters` that are aggregated using the given `method`. Parameters may either be literals, references to resource attributes or the result of other functions. SISL offers a basic set of built-in processing instructions. Though the set of available operations is kept small, a mechanism for future extensions is provided (see also Section V-B).

D. Notifications and conditions

The `notification` clauses (Figure 9) determine if and when (according to a `condition` clause) processed informa-

<table>
<tr><td>⟨notification⟩::=</td><td>[notification] ”notification{” ⟨condition⟩ ⟨declaration⟩ ”}”</td></tr>
<tr><td>⟨condition⟩::=</td><td>”condition{” ⟨identifier⟩ ⟨description⟩ ⟨boolExpression⟩ ”}”</td></tr>
<tr><td>⟨declaration⟩::=</td><td>”declaration{” ⟨valueset⟩ ”}”</td></tr>
</table>

Fig. 9. SISL Conditions

tion should be relayed. A `declaration` rule indicates whether single or multiple values should be submitted.

Conditions are given in the form of logical expressions in normal form (conjunctive or disjunctive normal forms are valid options). They can be constructed from binary predicate expressions (e.g. comparisons of values) or unary expressions when booleans are concerned. The common relational and arithmetic operators are supported, as for example equality, greater/less, AND, OR, XOR and so forth.

E. Example

To illustrate the usage of SISL, we employ an example derived from the scenario in Section II. Consider the computing service (see Figure 1, shaded area) that provides computing power originating from the underlying computing nodes and that is dependent on the DNS service and router in order to provide its service.

Thus, we define the available computing power to the user $c \in \mathbb{R}, 0 \leq c \leq 1$ of N computing nodes as follows:

$$c = D \cdot R \cdot \frac{\sum_{i=1}^{N} (p_i a_i)}{\sum_{i=1}^{N} p_i},$$

where $D \in \{0, 1\}$ and $R \in \{0, 1\}$ specify the availability of the DNS service and router respectively, p_i is a value describing the "strength" of computing node $i \in \{1, .., N\}$ and $a_i \in \{0, 1\}$ states the availability of computing node i. The values of p_i are out of scope of this work. Comparability of e.g. computing elements is a research area of its own. Representative work in the area of Grid accounting and billing includes [1]. The precise values of p_i are modelled as attributes of the resources and could be realised as database records.

The SISL representation of this example is shown in Figure 10. In lines 7 to 30 the resources and their attributes of interest are specified. Lines 31 to 59 present the functions executed by the adapters, while the functions executed by the service attribute factory are listed in lines 60 to 111. Finally, beginning at line 112, the condition under which a notification shall be passed on to the service management application is declared.

V. TOOL SUPPORT FOR SERVICE INFORMATION SYNTHESIS

As indicated in Section I, a basic requirement to an architecture supporting synthesis of service information is the reuse of existing data sources, such as already deployed management tools. Obviously, such data will be delivered in different formats. To interact with such tools, different APIs will have to be used, e.g. for requesting data or configuring monitoring options. Therefore, the consistent specification noted in SISL needs to be broken down into different "languages" according to whatever tools are used as data sources. In addition, the data received from all sources must be converted into a common syntax that can be used to describe the service.

A. Architecture overview

The Service Monitoring Architecture (SMONA) [3], [4] extended by a *Service Attribute Factory* (see Figure 11) component addresses these needs.

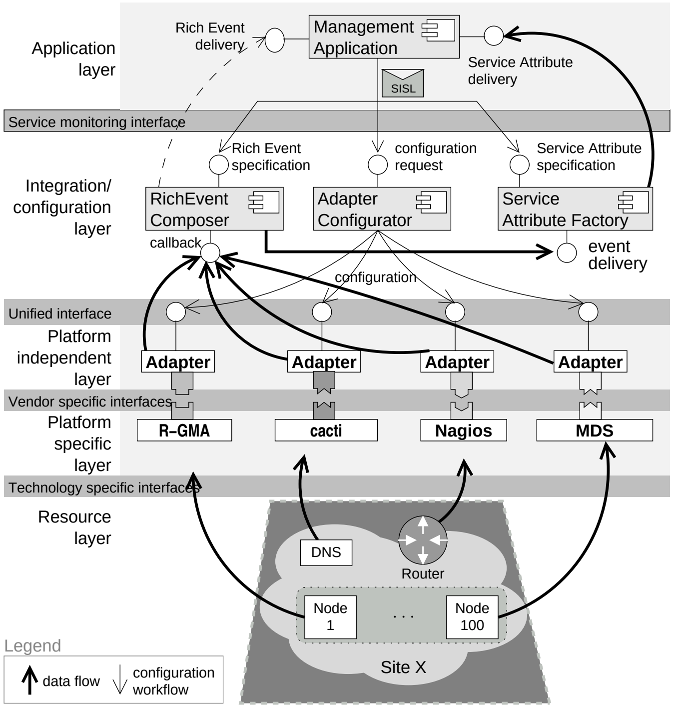

Fig. 11. An architecture for Service Attribute Synthesis

a) The resource layer: This layer encompasses infrastructure components, applications and other sources of "raw" data. The data available at this layer is specific to each resource/component, though some standardised interfaces and data formats may be available.

b) The platform specific layer: Resources are often managed by means of more or less specialised management tools (including scripts and "homemade" tools) that are found in the *platform specific layer*; they provide information pertaining to the infrastructure. Typically, information extracted from the resource layer will be processed and made available in a variety of formats.

c) Platform independent layer: To overcome the heterogeneity in the data sources at the platform independent level, *adapters* provide unification of the data format and basic configuration options. Every adapter needs to be capable of configuring the underlying resource (or tool) and of extracting the required data. The adapters present a common interface towards the higher layers.

The adapters' main task is to harvest the data and perform pre-processing as required by applicable `function` statements and deliver it in a common format to the *RichEvent Composer* component. Delivery is governed by conditions that pertain to temporal aspects, the method of delivery (push/pull) as well as conditions regarding other events in the data gathering process. Examples include the number of samples collected and thresholds for collected values.

d) Integration and configuration layer: To produce the service information required, as defined e.g. in a SISL document, adapters may need to be selected and configured

```
aggregation{
 id=compServAvailability
 description{
  author{DanciuFeldeSailer}
  date{20060901134242}
 }

/* parameters needed by adapters */
 resource{ id=siteA.DNS
  source{siteA.DNS}
  sourceAttrib{up_down}
  interval{1}
  return{boolean}
 }
 resource{ id=siteA.router
  source{siteA.router}
  sourceAttrib{up_down}
  interval{1}
  return{boolean}
 }
 resource{ id=siteA.node*
  source{siteA.node*}
  sourceAttrib{ up_down
   interval{1}
   return{integer}
  }
  sourceAttrib{ strength
   interval{0}
   return{float}
  }
 }

/* functions executed on adapters */
 function{ id=strength.node*.siteA
  description{
   text{strength of node}
  }
  method{mult}
  parameters{
   valueset{resourceRef{'strength
                @siteA.node*', 1}}
   valueset{resourceRef{'up_down
                @siteA.node*', 1}}
  }
  return{float}
 }
 notifications{
  condition{ id=nodeAvailability
   description{
    text{pass Availability of
           nodes to factory  }
   }
   timeout{60} // in seconds
  }
  declaration{
   valueset{functionRef{
         'strength.node*.siteA'}}
   valueset{resourceRef{
      'strength@siteA.node*', 1}}
  }
 }

/* functions executed by factory */
 function{ id=sumStrength
  description{
   text{calculate sum of strengths}
  }
  method{sum}
  parameters{
   valueset{resourceRef{'strength
                @siteA.node*', 1}}
  }
  return{float}
 }
 function{ id=sumAvailability
  description{
   text{calculates sum of nodes}
  }
  method{sum}
  parameters{
   valueset{functionRef{
           'strength.node*.siteA'}}
  }
  return{float}
 }
 function{ id=availability
  description{
   text{calculate ratio of values}
  }
  method{div}
  parameters{
   valueset{functionRef{
          'sumAvailability'}}
   valueset{functionRef{
          'sumStrength'}}
  }
  return{float}
 }
 function{ id=serviceAvailability
  description{
   text{currently available
          computing power  }
  }
  method{mult}
  parameters{
   valueset{functionRef{
          'availability'}}
   valueset{resourceRef{'up_down
          @siteA.DNS', 1}}
   valueset{resourceRef{'up_down
          @siteA.router', 1}}
  }
  return{float}
 }

 notifications{
  condition{ id=service-
          AvailabilityNotification
   description{
    text{Availability of service}
   }
   timeout{60} // in seconds
  }
  declaration{
   valueset{functionRef{
          'serviceAvailability'}}
  }
 }
}
```

Fig. 10. SISL example: Specifying a service management attribute of the exemplary computing service, see also Table I and Section IV-E

accordingly. The task of managing the adapters, including deployment, activation and configuration, is performed by the *Adapter Configurator* component. The configurator uses the `resource` (Figure 7) declarations to determine the appropriate adapters, instantiate and configure these according to the `interval`, `function` and `source` clauses (see Figures 7 and 8).

The *RichEvent Composer* performs the composition of data according to SISL specification. It gathers the (pre-processed) data from all adapters related to a service attribute and produces data records in accordance to `aggregation` (Figure 6) specifications. The `notification` (Figure 9) clauses determine when data records are compiled and dispatched. The resulting records (called *RichEvents*) can be made available to a management application (indicated by the dashed line in Figure 11) and/or relayed to other components. In practice, a middleware bus is used for transport of notifications.

e) Application layer: The architecture's clients found in this layer can be management applications or—later on—Service Agents that implement a Service MIB.

B. Synthesis of service attribute values

Typically, the synthesis of service attribute values requires further processing of RichEvents produced by the composer component. The *Service Attribute Factory* component applies the processing instructions expressed in `function` clauses (see Figure 8) on the RichEvent to produce an attribute value. It is equipped with an extensible library of mathematical operations and functions used to operate on the elements of service information contained in the RichEvent. Dynamic binding of operation names referenced in SISL documents to library functions is used to ensure flexibility in future extensions.

VI. Related Work

In this section, we present representative work in the area of service information modelling and service description languages and discuss its significance to the approach proposed in this paper.

a) CIM Metrics: The Common Information Model [6] introduced by the Distributed Management Task Force is an object-oriented information model that aims at providing a common way to represent information about networks and systems, as well as services. Within CIM, the *Metrics Model* connects to our work, in that it provides a means to express arbitrary metric information for all kinds of objects in the CIM class hierarchy, including the class *CIM_Service*. Keller et al. [13] describe an extension of the Metrics Model for metric aggregation, along with a CIM *Measurement Provider* that implements an aggregation service based on the model. This approach is built around the CIM framework; it assumes a CIM-based infrastructure model being in place and relies on CIM *Providers* for data harvesting.

b) Internet Information Model: The Internet Information Model (IIM) [16] designed by the Internet Engineering Task Force revolves traditionally around the Simple Network Management Protocol (SNMP). In this context, two MIBs have been devised that deal with aggregation of management information, namely the Distributed Management Expression MIB [17] and the Event MIB [12]. Both approaches combined allow to synthesise new MIB objects by performing mathematical operations on existing attributes, monitor these objects and specify conditions for triggering events (e.g. sending a notification).

Despite the structural similarity to the Service MIB approach, the RFCs inherently target the internet management domain. While isolated attempts to model single service classes [10] are part of the specification, IIM focus clearly lies on the network and systems management, where a plethora of MIBs have been specified. In that context, however, it constitutes a rich source of component parameters for synthesis of service attributes, exploitable by SNMP adaptors within SMONA (see Section V).

c) Shared Information/Data Model: As part of the New Generation Operations Systems and Software (NGOSS) program the TeleManagement Forum released SID (Shared Information/Data model) [21]. SID's strength clearly lies in its modelling of higher-level concepts (e.g. service , SLA), where most entities and attributes have been defined. While in this regard SID offers considerable benefits over IIM and CIM in terms of maturity, it currently shows deficits in expressing low-level details of resources (component parameters). However, SID's sound modelling of service and component interrelationships render it easy-to-apply for Step 3 of the methodology presented in this paper (Figure 5).

d) Service Modelling Language: Gopal's Service Modelling Language (SML) [9] allows services to be defined by selecting appropriate values in a 9-dimensional space, namely type, size, duration, connectivity, QoS parameters, protocol parameters, value added features, assurance, and pricing. Since these dimensions are used to describe services unambiguously, they can be seen as a sort of service attributes. SML features a set of primitives to combine or aggregate services, mainly by performing mathematical operations on the values in the 9-dimensional space. SML's focus on these nine dimensions for describing services attributes constrains its general applicability. However, it exhibits a number of sound language features, e.g. the use of patterns in declarations.

e) Web Services: Modelling and composition of services has been the subject of intense research in the area of Web Services. The Web Service Description Language (WSDL) [2] constitutes a XML-based language to describe how web-based services can be invoked. Its scope is similar to that of interface definition languages, such as CORBA's IDL. The OWL-S coalition has authored a markup language for describing the properties and capabilities of Web Services [19]. OWL-S is intended to be used by software agents for discovery and planning of services (e.g. generation of composite services in accordance with user goals). A principle underlying both languages is the abstraction of implementation details; they are agnostic regarding the components that form a service. However, they do not address the issue of binding component parameters to services.

f) Web Service Level Agreement Framework (WSLA): The WSLA Framework [14], [5] is targeted at defining and monitoring SLAs for Web Services. It consists of a formal language for SLA specification and a runtime architecture. The WSLA language allows developers to define three aspects of a SLA, namely parties, service description and obligations. The service description entails the definition of SLA parameters (e.g. *OverUtilization*) as well as instructions on how these parameters should be aggregated. Aggregation instructions can either be defined in terms of functions or measurement directives. The latter specifies how an individual parameter is to be measured, e.g. by providing an URI (Universal Resource Identifier), a protocol message or script execution. The specification of instructions is, however, imperative in nature and the binding to concrete resources is deferred to the implementation of the respective instruction. The WSLA language is intended to drive the configuration of the runtime architecture. It is implemented as a collection of Web Services including *SLA Establishment*, *SLA Deployment*, *SLA Measurement and Reporting* and *SLA Termination*.

g) RDF—Resource Description Framework: The World Wide Web Consortium's RDF [22] specifies a XML-based format for resource description in form of directed graphs. In RDF anything represented by a *Universal Resource Identifier (URI)* is regarded as a *resource*. Properties or attributes of these resources are described as RDF *description*s and can either be a *literal* or a pointer to another resource. A set of descriptions form the RDF graph, the resources and literals being the nodes and the properties forming the edges.

VII. Discussion

In this paper we have emphasised the importance of service attributes in the management of contemporary and future IT services. We have proposed a methodology for the specification of such attributes and presented SISL, a declarative language suited for formal attribute specification. To facilitate technical realisation of service attributes, we have proposed an architecture capable of synthesising attribute values from existing resource data. In this section, we will address a number of issues left for further study and discuss the limitations of the approach presented.

1) Language extensions: The aggregations performed in the Integration layer of SMONA (specifically in the Service Attribute Factory, see Figure 11) rely on a library of mathematical operations being present. The contents of this library effectively limit the power of expression offered by SISL. As there is virtually an infinite number of conceivable mathematical and statistical operations that can be performed on resource data, it is probable that users will eventually require operations not provided. The dynamic binding (by name) of the required library operations offers an implicit, generic extension mechanism. However, it is subject to uncontrolled growth of the library set. The most frequently used operations should be compiled into a "standard operations set".

2) Derivation of generic service attributes: The larger part of this paper is focused on the specification of service attributes with a given semantics. However, as suggested in Figure 5, attributes of IT services are selected or derived from the overall management paradigm used and the high-level management requirements imposed onto the IT organisation. Three sources of service attributes are discussed in short in this section: process-oriented IT Service Management (ITSM) frameworks, the classic functional areas of management and the IT management alignment to the (formalised) needs of business. For the assembly of a Service MIB domain limits as well as the life-cycle of IT services must be taken into account.

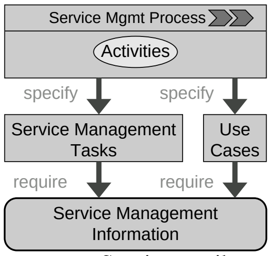

Fig. 12. Service attributes originating from ITSM procedures

a) Taking into consideration ITSM frameworks: The increasing use of ITSM frameworks like ITIL's Service Support [18] suggests that such frameworks constitute a source of requirements with regard to service attributes [20]. The main goal of such process-oriented frameworks is to control—under business aspects—the life-cycle of services provided in an IT organisation. As shown in Figure 12, the introduction of management processes implies the specification of management tasks and use-cases that must be executed manually or in reliance on service management tools. In either case, managers have no direct interest in the current operative state of infrastructure elements. The service related information they do need can be specified as a set of service attributes.

b) The functional areas of management: The classic OSI functional areas (FCAPS) can be leveraged as a guideline to service management [7]. Every FCAPS area has specific needs for information regarding a management target. These needs could be met by creating suitable service attributes—or by providing templates as described later on in this section. Such attributes should support the management tasks defined for the functional areas they are derived from.

c) Business-driven technical service management: Management requirements derived from business needs can be acquired from SLAs, OLAs or similar contracts. These should reflect customer demands as well as the impact of service failures on the business. As such, they present requirements on the management information (i.e. service attributes) available with respect to the services offered. For this task, however, specialised approaches (such as WSLA) already exist.

3) Assessing the impact of device failure: The aggregations specified for service attributes allow reasoning regarding the "importance" of a device. Taking a closer look at the aggregating function in our example (see section IV-E), it is obvious that failures of the DNS service *or* the router will cause a failure of the service. This identifies those resources as single points of failure.

The adaptation of aggregation rules along the service life-cycle raises additional interesting questions. In particular, automation support for adjusting an aggregation function in response to changes in the service provisioning constitutes a challenging issue.

4) Inter-domain and Grid management: Initially, SISL has been designed to specify service attributes in single domain setups. As one next step, the language shall be enhanced to work in multi-domain and Grid environments. Therefore, several requirements beyond the ones mentioned in Section II have to be taken into account, especially security considerations like authorisation, authentication, data integrity and confidentiality, performance considerations (e.g. delay), clock synchronisation and the enforcement of information sharing and privacy policies. Further, the application of SISL and the before mentioned service monitoring architecture will be deployed in a Grid environment. Thus, VO (Virtual Organisation) Management Systems have to be interfaced. This leads to the necessity to respect both the policies *of* VOs and the existing policies *applicable to* VOs. Additionally, highly dynamic service composition has to be supported as a typical Grid characteristic, implying highly dynamic resource allocations and possibly short-lived VOs.

5) Service templates: Inherently, the methodology presented in Section III relies on manual execution. It is tied to expert knowledge in that scenario-specific characteristics need to be accommodated. Since services can be provisioned in a wide variety of ways—varying across different vendors, technologies, and product offerings— this seems unlikely to change. As an alleviation of this problem, we are working on a template library for standard services. These templates are intended to provide a basic set of attributes and aggregation rules that can be adapted—and consequently refined—to match a specific scenario. Towards this goal, we analyse common application domains in order to identify invariant service characteristics. For instance, we assume a standard web hosting service to be composed of a web server, middleware server, and database, as well as a router. Based on this simple model, a number of generic service attributes and aggregations can be derived, e.g. that the connectivity of the web hosting server depends on both the connection of the employed servers and the router. Although a serious effort is obviously required to build a comprehensive template library, related work [15], [7] shows that a template-based approach to describing services is indeed feasible.

6) Application in distributed security scenarios: As future work, the application of SISL and SMONA in the context of Grid security management is planned. We are trying to leverage our approach to support development of a Grid intrusion detection and reporting system while taking the Grid-typical issues mentioned in VII-.4 into account.

7) Implementation: Up to now, most of our implementation efforts have concentrated on the platform specific and platform independent layer of SMONA (see section V). In this context, we have developed several adaptors, including an *nagios* bridging adapter as well as an *iptables* adapter. Communication with upper layers of SMONA has been facilitated using the CORBA middleware. Since a number of CORBA bindings for programming languages exist, this also introduces flexibility in adapter development – with adaptors today being implemented in C++ or JAVA. While the basic functionalities (such as parsing SISL documents or basic aggregation functionalities) of the integration and configuration layer have been realised, the development of a graphical user interface would further ease the application of SMONA. This is being addressed by future work, together with an extended mathematical library for aggregation functions.

ACKNOWLEDGEMENT

The authors wish to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of this paper. The MNM Team founded by Prof. Dr. Heinz-Gerd Hegering is a group of researchers of the University of Munich, the Munich University of Technology, the University of Federal Armed Forces Munich and the Leibniz Supercomputing Centre of the Bavarian Academy of Sciences. Its web-server is located at http://www.mnm-team.org. [1]

REFERENCES

[1] Alexander Barmouta and Rajkumar Buyya. GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration. *ipdps*, 00:245a, 2003.

[2] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. Technical report, W3C, March 2001. W3C Note.

[3] V. Danciu, A. Hanemann, H.-G. Hegering, and M. Sailer. IT Service Management: Getting the View. In E. M. Kern, H.-G. Hegering, and B. Brügge, editors, *Managing Development and Application of Digital Technologies*. Springer Verlag, June 2006.

[4] V. Danciu and M. Sailer. A monitoring architecture supporting service management data composition. In *Proceedings of the 12th Annual Workshop of HP OpenView University Association*, number 972–9171–48–3, pages 393–396, Porto, Portugal, July 2005. HP.

[5] Markus Debusmann and Alexander Keller. SLA-driven Management of Distributed Systems using the Common Information Model. In *Integrated Network Management VII, Managing It All, IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003)*, volume 246, Colorado Springs, USA, March 2003. IFIP/IEEE, Kluwer Academic Publishers.

[6] Distributed Management Task Force (DMTF). Common Information Model (CIM) Version 2.9. Specification, June 2005.

[7] G. Dreo Rodosek. A Generic Model for IT Services and Service Management. In *Integrated Network Management VII, Managing It All, IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003)*, volume 246, pages 171–184, Colorado Springs, USA, March 2003. IFIP/IEEE, Kluwer Academic Publishers.

[8] M. Garschhammer, R. Hauck, B. Kempter, I. Radisic, H. Roelle, and H. Schmidt. The MNM Service Model — Refined Views on Generic Service Management. *Journal of Communications and Networks*, 3(4):297–306, December 2001.

[9] Rajeev Gopal. Unifying Network Configuration and Service Assurance with a Service Modeling Language. In R. Stadler and Ulema M., editors, *Proceedings of the 8th International IFIP/IEEE Network Operati ons and Management Symposium (NOMS 2002)*, pages 711–725, Florence, Italy, April 2002. IFIP/IEEE, IEEE Publishing.

[10] H. Hazewinkel, C. Kalbfleisch, and J. Schoenwaelder. RFC 2594: Definitions of managed objects for www services. RFC, IETF, May 1999.

[11] H.-G. Hegering, S. Abeck, and B. Neumair. *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999.

[12] R. Kavasseri. RFC 2981: Event mib. RFC, IETF, October 2000.

[13] A. Keller, O. Benke, M. Debusmann, A. Köppel, H.M. Kreger, A. Maier, and K. Schopmeyer. The CIM Metrics Model: Introducing Flexible Data Collection and Aggregation for Performance Management in CIM. *IEEE eTransactions on Network and Service Management (eTNSM)*, 1(2), December 2004.

[14] Alexander Keller and Heiko Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management, Special Issue on "E-Business Management"*, 11(1), 2003.

[15] Pankay K.Garg, Martin Griss, and Vijay Machiraju. Auto-Discovering Configurations for Service Management. *Journal of Network and Systems Management*, 11(2):217–239, 2003.

[16] K. McCloghrie, D. Perkins, and J. Schoenwaelder. RFC 2578: Structure of management information version 2 (smiv2). RFC, IETF, April 1999.

[17] R. Kavasseri (Ed. of this version). RFC 2982: Distributed management expression mib. RFC, IETF, October 2000.

[18] Office of Government Commerce (OGC), editor. *Service Support*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2000.

[19] OWL-S Coalition. OWL-S 1.1 release. http://www.daml.org/services/daml-s/0.9/.

[20] M. Sailer. Towards a Service Management Information Base. In *IBM PhD Student Symposium at ICSOC05*, Amsterdam, Netherlands, December 2005.

[21] TeleManagementForum. Shared Information/Data (SID) Model Concepts, Principles, and Domains. Technical report, July 2006. GB 922.

[22] World Wide Web Consortium W3C. Resource Description Framework (RDF), February 2004. http://www.w3.org/RDF/.

[1]Parts of this work have been funded by the German D-Grid Initiative under contract 01 AK 800 B.