SPECIFICATION OF DEPENDENCIES FOR IT SERVICE FAULT MANAGEMENT

Andreas Hanemann, David Schmitz
Munich Network Management Team, Leibniz Supercomputing Center
Boltzmannstr. 1, D-85748 Garching, Germany

Patricia Marcu, Martin Sailer
Munich Network Management Team, University of Munich (LMU)
Oettingenstr. 67, D-80538 Munich, Germany

Keywords: Dependency modeling, service management, event correlation, impact analysis.

Abstract: The provisioning of IT services is often based on a variety of resources and underlying services. To deal with this complexity the dependencies between these elements have to be well-known. In particular, dependencies are needed for tracking a failure in a higher-level service being offered to customers down to the provisioning infrastructure. Another usage of dependencies is the impact estimation of an assumed or actual resource failure onto the services to allow for decision making about appropriate measures.
Starting from a set of requirements an analysis of the state-of-the-art shows the contributions and limitations of existing research approaches and industry tools for a configuration management solution with respect to the dependencies. To improve the current situation a methodology is proposed to model dependencies for given service provisioning scenarios. The proposed dependency modeling is part of a larger solution for an overall service management information repository.

1 INTRODUCTION

In a competitive environment, service providers strive to deliver services in a lean and agile manner, despite the rising complexity and heterogeneity within the network. The advent of new technologies such as virtualization of hardware and service-oriented architectures add to this complexity and thus even increase the challenge for IT management.

One of the key challenges for service providers is efficient fault management. When a customer experiences problems with a service, the provider needs to react quickly in order to honor the Service Level Agreements (SLA) in effect between provider and customer. Conversely, it is desirable to determine the impact onto services and Service Level Agreements when problems with resources or subservices are detected (Hanemann et al., 2005).

Fault Management requires the assessment of dependencies of a service on specific resources, resource interdependencies as well as the mapping of resources on the services provided. This is to infer the actual cause of service problems, to identify the malfunctioning or misconfigured resources, and to devise a problem solution. This process is by no means trivial. In large scale systems, it is nearly impossible without appropriate tool support. Automation support, however, requires dependencies to be formalized in a machine-readable format. Service providers therefore maintain management models, which essentially represent an abstraction of their IT infrastructure. However, state-of-the-art management models show deficits regarding the specification of dependencies and thus provide only rudimentary support with respect to the needs of IT service fault management.

In this paper we examine how dependencies can be modeled in a systematic and service-oriented way that allows us to extend – and thereby refine – current management models. In Section 2 a set of requirements for the dependency modeling is presented which is used to analyze related work in Section 3. Our approach for specifying the dependencies based on the analysis of interactions is detailed in Section 4. Conclusion and future work can be found in the last section highlighting the inclusion of this work into an approach for management information modeling.

2 REQUIREMENTS

In Fig. 1 different aspects which have to be considered for the modeling of dependencies are depicted. These

Hanemann A., Schmitz D., Marcu P. and Sailer M. (2006).
SPECIFICATION OF DEPENDENCIES FOR IT SERVICE FAULT MANAGEMENT.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 257-260
DOI: 10.5220/0001319502570260

are explained in the following.

The *type of dependency* is used to distinguish between inter-service, service-resource, and inter-resource dependencies. These denote dependencies between services, services and resources as well as between resources, respectively.

A distinction can be made for the *level of detail* for services since a service can usually be divided into service functionalities. For a web hosting service it is e.g. possible to retrieve web pages or to fill out and transmit an online form. Dependencies can therefore be detailed whether they are relevant for a specific service functionality.

An important characteristic of dependencies is whether they can be treated on their own (isolated) or whether they involve multiple resources or services which is usually the case for *redundancies*. A service may use a set of redundant hardware components. Therefore, an unavailability of a server has to be regarded in conjunction with the other servers.

Another important characteristic is the *dynamic* which denotes the treatment of changes in the dependencies. Especially for fault management it is important to know whether it can be assumed that the dependencies remain unchanged during the fault diagnosis.

The *service lifecycle* describes the change of dependencies from the initial installation of the service until its removal.

The *degree of formalization* of the dependencies also needs to be considered. Dependencies cannot only be formalized or specified as pseudocode but can also be defined as formal model.

Finally, not only functional dependencies can be taken into account, but also those resulting from organizations (dimension *characteristic*). However, this paper concentrates only on the functional dependencies.

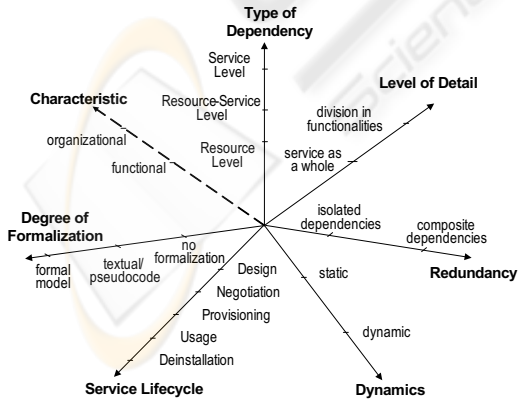

Figure 1: Requirement dimensions.

Dependencies play an important role in the area of IT service fault management. In particular, they are a prerequisite for the following two use cases: For fault diagnosis the dependencies should be traversable in a top-down manner to identify a resource as root cause of a service problem, while the opposite direction is relevant for impact analysis to estimate the impact of a resource failure for services.

3 RELATED WORK

In the analysis of related work we distinguish between the identification, modeling, and use of dependencies.

In the literature the term “dependency” is often used without a precise definition. Examples are the definition of absent, weak, medium and strong dependencies in (Bagchi et al., 2001) or the distinction between “and” and “or” dependencies in (Rodosek and Kaiser, 1997), which did not provide any further attributes.

Caswell and Ramanathan (Caswell and Ramanathan, 1999) describe dependencies for services, especially for ISPs. They define five kinds of dependencies which are partially similar with our definition of terms. They specify inter-service dependencies in the way it is done in the previous section. Service-resource dependencies in our terms can be seen as an superclass of the link, component, and execution dependencies in this model. The execution dependency reflects the dynamic in service provisioning where the execution of a functionality can be done using a selection of resources. The organizational dependency could be covered by the definition of sub-services. However, the specification does not go into further details and is tied to ISPs.

Common Information Model (CIM) CIM (DMTF, 2005) is an object-oriented information model that aims at providing a common way to represent information about networks and systems as well as services. Dependencies between CIM elements are modeled as association classes and are derived from a single base class *CIM_Dependency*; the inheritance hierarchy is therefore flat. CIM would particularly profit from a more systematic approach that allows to arrange the existing dependencies in a hierarchical structure.

NGOSS SID The Shared Information/Data (SID) model (TeleManagementForum, 2003) is part of the initiative Next Generation Operation Support and Software (NGOSS). It employs an object-oriented modeling approach and constitutes a framework for defining service management information from a business-oriented point of view. Since SID is currently in an early development stage, work has mainly focused on providing the major building blocks of the

model. Albeit this did not include systematic modeling of dependencies in terms of hierarchies, it can be expected for upcoming versions of SID.

In sum, it can be concluded that dependency modeling cannot be regarded as a solved research issue with respect to the requirements of service management.

4 DEPENDENCY SPECIFICATION METHODOLOGY

In this section a generic model for the dependency specification is developed striving to fulfill the requirements delineated in Section 2. We also show, how the model helps in addressing the service fault diagnosis.

Dependency Modeling Dependencies which are encountered in practice usually form a hierarchy which results from the composition of resources and subservices into more complex higher-level services. Therefore, the structural design pattern *composite* is used as basis for the modeling.

Figure 2 depicts the proposed dependency model. The object *Dependency* reflects the *component* object of the pattern and declares the interface for objects in the composition *CompositeDependency* (the *composite* object).

Some attributes characterize the *Dependency* class: *antecedent* has the type *ServiceBuildingBlock* (abstract superclass of services and resources) and is the antecedent, *dependent* represents the dependent of a dependency also having the type *ServiceBuildingBlock*. The *strength* attribute (string) represents the importance of the dependency with the values *weak, medium, strong, mandatory* which means that is describes the redundancy relation. The last attribute *statusLifeCycle* is an indicator for the state of the dependency along the service lifecycle using the values *planned, active, inactive, withdrawn*.

Usually, a service exhibits a set of functionalities. These functionalities can in turn be divided into subfunctionalities and can represent the whole service (global functionality) as well as a single functionality or a partial functionality of the service. This kind of modeling allows to find a trade-off between the modeling effort and its benefit in a given scenario.

A set of operations are defined: the *add* operation adds a new dependency to the container, whereas the *remove* operation removes a dependency from the container. The operation *getChild* returns the n-th child of the dependency, *getChildList* lists all the direct children. The operations *getDependent, getAntecedent, testPresence,* and *getImpact* are designed for fault management (see below).

The leaves of the composite pattern are formed by the three kinds of dependencies. *Inter-Resource-Dep* represents inter-resource dependencies with the inherited attributes *strength, antecedent, dependent, statusLifeCycle* defined as in the component class Dependency. The type of the attributes *antecedent* and *dependent* is Resource. The operations will implement the abstract operations from the class Dependency.

The *Service-Resource-Dep* class represents the service-resource dependency and has also the attributes and operations inherited from class Dependency. Besides of the different implementation of operations, the attribute *antecedent* has the type Resource and the attribute *dependent* the type Functionality.

Inter-service dependencies are formalized as instances of the *Inter-Service-Dep* class which has the same attributes and implements the same operations as class Dependency. The type of the attributes *antecedent* and *dependent* is Functionality. This class contains many hierarchically structured instances resulting from the functionality definition. Dependencies can exist between two services, between a service and a service functionality (i.e. one service is decomposed into its functionalities, while this decomposition is not made for the other service), or between two service functionalities.

The composite class in this pattern is the class *CompositeDependency*. It defines behavior for dependencies having children, it stores this children (leaves or composite components) and implements operations relating on child component (*add, remove, getChild, getChildList*) in the *Dependency* interface.

Methods for Fault Management Application In addition to the child oriented operations there are operations which facilitate the navigation in a system and which are useful for fault diagnosis and/or for impact analysis.

The operation *getDependent* returns all the ServiceBuildingBlock objects which are dependent from the actual dependency and *getAntecedent* returns all the ServiceBuildingBlock objects the actual dependency depends on. Depending on the element in which it occurs (leaves or composite elements) it has another return value (functionality or resource). The operation *testPresence* returns the status of the dependency for a given point in time representing whether the dependency was in effect at that moment (as a boolean).

5 CONCLUSION

We have presented a systematic modeling approach that allows the specification of complex, dynamic and

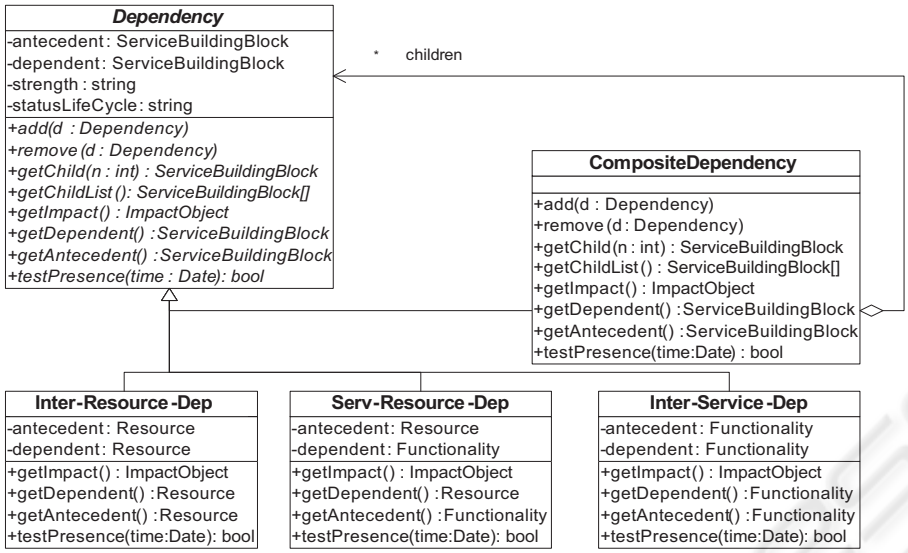

Figure 2: Dependency Model.

adaptable dependencies. While state-of-the-art management models are tackling these issues in different ways, they would benefit from the systematic approach presented in this paper.

We are currently implementing our approach by extending a specific management model, namely DMTF's CIM and will apply it for the management of IT services at the Leibniz Supercomputing Center in Munich. We also plan to merge the findings of this paper with the Service Management Information Base approach (Sailer, 2005), a research project conducted within our research group.

ACKNOWLEDGEMENTS

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 directed by Prof. Dr. Heinz-Gerd Hegering is a group of researchers of the University of Munich, the Munich University of Technology, the University of the Federal Armed Forces Munich, and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences. Their web–server is located at *http://www.mnm-team.org*.

This paper was supported in part by the EC IST-EMANICS Network of Excellence (#26854).

REFERENCES

Bagchi, S., Kar, G., and Hellerstein, J. (2001). Dependency analysis in distributed systems using fault injections: Application to problem determination in an e-commerce environment. In *Proceedings of the 12th International IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM 2001)*, Nancy, France.

Caswell, D. and Ramanathan, S. (1999). Using service models for management of internet services. In *HP Technical Report HPL-1999-43, HP Laboratories*, Palo Alto, California, USA.

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

Hanemann, A., Sailer, M., and Schmitz, D. (2005). A framework for failure impact analysis and recovery with respect to service level agreements. In *Proceedings of the IEEE International Conference on Services Computing (SCC 2005)*, Orlando, Florida, USA. IEEE.

Rodosek, G. D. and Kaiser, T. (1997). Determining the availability of distributed applications. In *Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management (IM 1997)*, pages 207–218, San Diego, California, USA. IFIP/IEEE.

Sailer, M. (2005). Towards a service management information base. In *Proceedings of the IBM PhD Student Symposium at the 3rd International Conference on Service-Oriented Computing (ICSOC 2005); IBM Research Report*, Amsterdam, The Netherlands.

TeleManagementForum (2003). Shared Information/Data (SID) Model Concepts, Principles, and Domains. Technical Report GB 922 Member Evaluation Version 3.1.