Derivation of Backup Service Management Applications from Service and System Models

Ingo Lück [1], Marcus Schönbach [1], Arnulf Mester [1], Heiko Krumm [2]

[1] Dr. Materna GmbH, Voßkuhle 37, D-44141 Dortmund, Germany
{Ingo.Lueck|Marcus.Schoenbach|Arnulf.Mester}@materna.de
[2] FB Informatik, LS IV, Universität Dortmund, D-44221 Dortmund, Germany
krumm@cs.uni-dortmund.de

Abstract. The backup of large data sets is preferably performed automatically outside of regular working hours. In highly structured computer networks, however, faults and exceptions may relatively frequently occur resulting in unsuccessful subprocesses. Therefore automated fault and configuration management is of interest. We report on a corresponding management system. Besides of monitoring and information provision it performs automated fault analysis and recovery functions under extension of the service management approach to the function-oriented management of information processing services. Moreover, it is model-based. An interactively constructed object-oriented model specifies management objectives and represents dependencies between the backup service provided and the services used. Moreover, the model is input to the derivation of the management application code. Thus, the combination of service management and modeling supports the productive development of automated management applications. The system is implemented on the basis of the Java Dynamic Management Kit and performs the management of a commercial network backup system in a heterogeneous environment.

Keywords. model-based management, IT-service management, derivation of management systems, model-based development of management systems

1. Introduction

The notion of service management firstly occurred in the field of telecommunication network management as an answer to the growing complexity of networks and the user demand for a broad and flexible spectrum of high-quality services. It introduces an abstract and user-oriented view which complements the traditional resource-centered management and provides a common and technology-independent context for component management [Hal96]. Meanwhile, not only telecommunication but also information processing organizations were subject to a shift from technology provision to service provision. The users now start to demand more than technology-dependent best-effort support of few applications. The services of applications and their availability, performance, security, and reliability needs shall be addressed by so-called service level agreements. The business users want to rely upon application services as tools supporting their mission within the enterprise. The information processing organization shall achieve and maintain the agreed service levels by means of service-oriented management measures forming the so-called IT-service management [Mcb98] which presently can be supported by service monitoring, reporting, and capacity planning tools (e.g., [Haw98]).

Future application management systems, however, shall have a high degree of automation and shall preferably perform more than only monitoring and performance management operations. The notion of IT-services complies with both requirements. IT-services define abstract and user-oriented objectives of automated management which are not restricted to quantity-oriented operations for accounting, performance management, and quality of service control but also can guide the more function-oriented operations of configuration, fault, and security management. Nevertheless, function-oriented management operations are relatively specific. They mostly depend not only on the services provided and on the services used but also to a certain extent on the available resources and the dependencies between resources, services, exceptions, and faults. Due to the broad variety of business-near application services, therefore, an extensive set of tailored components for the automated management is needed and special support for the productive management component development is of interest.

Our work represents one promising step into the direction of productive development of management systems for automated and function-rich management of IT-services. It consists of two parts. In principle, we propose a model-based development approach. In a practical case study, we applied the approach to an enterprise-wide backup service. The model-based development starts with an abstract model of the services to be provided which is directly oriented at the job definitions of the backup service system. In fact, the initial model is mechanically derived from the job definitions. Thereafter, it is interactively refined by the introduction of the services used, of the resources, and of the dependencies between. The resulting detailed model is finally subject to the generation of management components. In this way, a powerful design and development support is realizable which moreover may in the future be improved by supplying pre-defined generic model elements. The practical case study achieves the automated fault and configuration management of enterprise-wide distributed and heterogeneous backup services (moreover, performance management and capacity planning functions are integrated). We chose backup services, since on the one hand, of course, backup services are more basic and less specific than business-oriented applications. They therefore seemed to be more appropriate for experimentation with a new approach. On the other hand there was a real needs to automate the operation of nightly backup processes which have to copy large data sets in a highly structured network under the presence of faults and exceptions.

In more detail, backups are performed by a commercial network-based backup system [Leg97]. The management system is distributed and based on the Java Dynamic Management Kit J-DMK [Jdm98]. Management agents reside in the client and server hosts of the backup system. They forward notifications and operating system information to the manager. Moreover, they can influence and restart components of the backup service and of the used services. The manager – derived from the model definition by a modeler tool – installs and configures the agents by means of J-DMK-functions. It communicates with its agents via events and remote method invocations. Before the establishment of this management system problems occurred which caused incomplete operation of backup processes. Mostly used services (e.g., domain name servers) were not available. Moreover, sometimes backup demons on client hosts had to be restarted. Now the introduced manager monitors the state of the necessary system services. It reacts on problem notifications by automated recovery of used services, by restarts of backup demons, and by repetitions of backup processes leading to a decrease of incomplete backups. Moreover, it provides detailed and pre-processed information for non-automated fault-management measures.

Our model-based approach for the definition of management objectives and for the derivation of the manager code is related to several existing approaches. So, in the sequel we first give an overview over the usage of models for management systems. Thereafter we outline the backup service to be managed in more detail. The next section presents the modeling and is followed by a description of the production of the manager. Finally we sketch the implemented system and note down concluding remarks.

2. Model-Based Management

Meanwhile many management approaches are based on explicit model notions resulting in a variety of modeling techniques and applications. In order to relate our approach to this existing work, we provide a short general classification of model-based management. Moreover this gives an overview and may help to identify further directions of model applications. Before, we shortly outline the use of models in the field of general software development which basically influences the design of management systems.

As Booch, Rumbaugh, and Jacobson point out in connection with their proposal of the unified modeling language UML [Uml97], models simplify reality and support the concentration on essentials, thus facilitating understanding and design. They devote UML to the modeling of systems under development. Several diagram types correspond to submodel types serving for the description of different aspects of structures and behaviors with the objective of visualizing, specifying, constructing, and documenting designs. Besides of UML, other models, like workflow process models [War94], development process models [Gra95], data base and information system models [Ere92], communication protocol and process behavior models [Tur93], as well as very specialized models of certain application-domains (e.g., system theoretic models of control loops in real-time systems) are in use. In general, models may reflect aspects of system development at all or of systems and their embedding on their own describing requirements, system structures, data schemes, system architectures, or functions and behaviors. They may aim at description and documentation supporting the understanding of humans, at conception of machine-processable implementation parts, or at specification of man-machine interfaces. Models may also exceed description and provide for explicit means of human or mechanical reasoning. They may be in use at design time, at compile / application generation time, or at runtime.

In the field of automated management, models may serve for specialized purposes and may concentrate on special functional areas (e.g., FCAPS, change or service management). Models may support typical development tasks (e.g., system analysis resulting in requirements and policies as well as design and implementation resulting in specifications, code modules, administration and modification schemes) or direct operation by maintaining structured system information and providing corresponding notions (e.g., for representation and processing of monitoring and audit information or for documentation and handling of changes or troubles). Models may be related to integral views of a system as a whole (e.g., the sum of service provisions) or they may focus on components like the managed system, the management system, or parts of them (e.g., certain managed objects, some agents). They may concentrate on static structures, on dynamic functional behavior, or on performance and quality of service properties. On a high abstraction level models may reflect corporate strategies. On a lower level tactical relations of certain management domains may be modeled, while more detailed models may focus on operational elements.

From the utilization point of view, following modes of model application are of main interest:

- *Specifications for system development*
  Static information models of managed objects define the scheme of the management information base MIB. This early and widely applied approach [Mib91, Gdm92] is the starting point of many extensions and improvements. So, extended MIBs can serve as basis for the definition of management policies [Wie94]. The more recent approach of the common information model CIM supports unified views for the integral management of complex and heterogeneous IT-environments [Cim98].
- *Formal specifications*
  Formal models as they are described by formal specifications can support rigorous design analysis and verification. Moreover the models can be used for the generation of test cases (e.g., based on Z-specifications of managed objects [Fer97]).
- *Design by model refinement*
  The stepwise refinement of models can coincide with the refining design of system parts (e.g., refining transformation of policies in [Wie95]).
- *Analysis and Checks*
  Models and model-based reasoning techniques can support the analysis of management systems under development (e.g., detection of conflicts in policy sets [Lup97]).
- *Simulations of system elements*
  Model-based simulations can substitute parts of a real system in order to support early tests [Lun96].
- *Prototypes and animations*
  Besides of simulation for the purpose of testing, animated man-machine interfaces and prototypical function implementations may be of interest in order to support communication with customers and staff training.
- *System generation*
  Implementation elements can be generated from model definitions. So, [Enj96] and [Hei96] derive code from models statically. Furthermore, the common MIB-compiler tools translating information schemes into code stubs have to be mentioned here.
- *Runtime functions*
  Explicit model representations which are interpreted at runtime can contribute to the implementation of management functions. [Sab97] translates constraint models into code of diagnostic functions for network fault management. [Ohs97] applies

descriptions of event models for automated event correlation. [Kae97] traverses fault propagation models at runtime in order correlate events and isolate faults. [Rod97] calculates the availability of applications from attributed service dependency graphs.

- *Adaptive models*
  Model adaptation at runtime – well-known in real-time control systems – may also be of interest for future management systems. On one hand, model adaptations can reflect system changes, on the other, the model quality may be enhanced by model improvements which utilize runtime feedback information.

Combinations of different modes of model application may be possible. So our approach for model-based backup service management constructs an object-oriented model at design time which serves for the specification of management objectives and for the description of the managed system. Moreover, at compile time application code is derived from this model. Like [Rod97] the model describes the service provided, the different services used, and the dependencies between. As in [Kae97], the model information supports the localization of faults and the selection of appropriate recovery procedures.

3. Backup Service and Management

A backup-service is a client / server - architecture based service, that is responsible for saving and restoring local and remote data in a distributed environment.

There are two basic backup mechanisms. A full backup is a complete copy of all the data to be saved, whereas an incremental backup copies that data only that have been changed since the last backup process. In general, backup copies can be produced with or without compression and with or without verification of the copied data on the backup medium (e.g., tape, optical disk). A schedule lays down when which type of backup copy has to be made. If restoration of data is needed the time it takes depends on the type of backup copies that are available. Restoring from a full backup is relatively fast in contrast to the case where several incremental backup sets have to be taken into consideration. A backup application is often a distributed program. On the one hand there can be many different hosts (backup-clients in this context) that do not have a backup device of their own but manage data to be saved. On the other hand there can exist a series of backup-servers that receive client data over the net and save it to connected backup devices.

The backup-service consists of two parts, one part creates the backups, the other performs restoration. Quality parameters are reliability, backup creation speed, restore speed, and complete automation (e.g., no manual change of backup media should be needed). Complete automation is especially important for the backup creation, because this is often done outside of regular working hours – mostly at night – when there is minimal other load on data network and hosts. The success of all actions of the backup-service should be documented in a log-file with corresponding reasons if unsuccessful. Furthermore the backup-service should support heterogeneous environments and should be scaleable.

There are various services used by the backup-system. A backup-server uses backup-devices as well as the CPU, memory and the hard disks of its host system.

Domain name servers are used to resolve IP-addresses. The network is used to transfer control and backup data. On a client-host the CPU and memory are used by the backup-client process (e.g., for data compression). The hard disk may be used for a local index-database. The quality of these services (e.g., CPU usage, memory usage, domain name server availability, network throughput and error rate) influences the quality of the backup-service.

Legato-Networker, the backup system we use in our implementation, knows four types of physical components: backup-devices, servers, interfaces (network-interfaces that are available in server-hosts) and clients [Leg97]. Additionally, there are some logical elements supporting definition and management of backup-services. Backup-groups are used to form groups of clients with comparable backup requirements. Schedules are assigned to single clients or to save-groups. They document at what time or period what type of backup should be created. Label-templates define patterns for the generation of names (labels) for backup-media. Volume pools allow the assignment of backup-data (determined by backup-group, client, and / or backup-type) to particular backup-media and document if entries to the client's index-databases should be made. Policies mainly determine how long data in the media- and index-database are stored. Moreover, directives can state special procedures for particular files. Finally event notifications have to be mentioned. They are generated to transmit state and error messages to specified destinations.

For the management of the Legato-Networker backup service we identified the following functional requirements: fault detection, isolation and logs, fault tolerance measures (backup process recovery, reconfiguration of used services), performance monitoring and capacity planning, as well as convenient graphical presentations of log information (events, event reactions, performance and status data).

4. Backup System Model

The model has to meet the following list of requirements. Appropriate representation of service level agreement is needed. Additionally the model has to support monitoring of the quality level at runtime. By that, assessment of service quality is enabled. Documentation is another important task. So, documentation of the configuration has to be supported for the backup service provided (e.g., save-groups, schedules, policies) as well as for the services used (e.g., which domain name server is used by a host). Furthermore relations between service components among themselves (e.g., schedules related to save-groups) and relations between service components and services used (e.g., assignment of backup-servers to hosts) have to be modeled. The fundamental requirement on the model is that the management application can be derived from it automatically. Finally fault detection, isolation and recovery has to be supported at runtime (e.g., if address resolution fails the corresponding domain name server has to be checked and possibly be restarted).

In order to meet this broad spectrum of requirements we use an object-oriented model, since even highly-structured domains can be represented in an easy-to-survey and flexibly extendable way. Moreover, dependencies and links can be represented by associations and runtime functions can be introduced by method implementations in a straight-forward way. The object classes used in our model are grouped in three packages as shown in Fig. 1 The *Service Model* package contains a class for every logical element supporting definition and management of backup-services. Additionally there are some classes that are used for service quality assessment. The *Service Level Agreement* class uses *Assessment Scales* to map service quality onto comparable states (i.e., OK, warning, critical). The physical components known by the backup-system are represented by classes in the *System Model* package. For interfaces and devices *Performance Measurements* are taken that can be viewed by administrators of the backup-services for analysis purpose. The *Services Used* package contains in addition to the *Services* themselves also *Service Configurations* (e.g., the amount of memory available in a host), *Service States* (e.g., memory usage for a host) and *Fault Detection Logs* (if for example a domain name server has to be restarted this action is documented). Except for the *Services* of the *Services Used* package all classes of the model are represented by Java classes. So, a model is built of Java object instances.

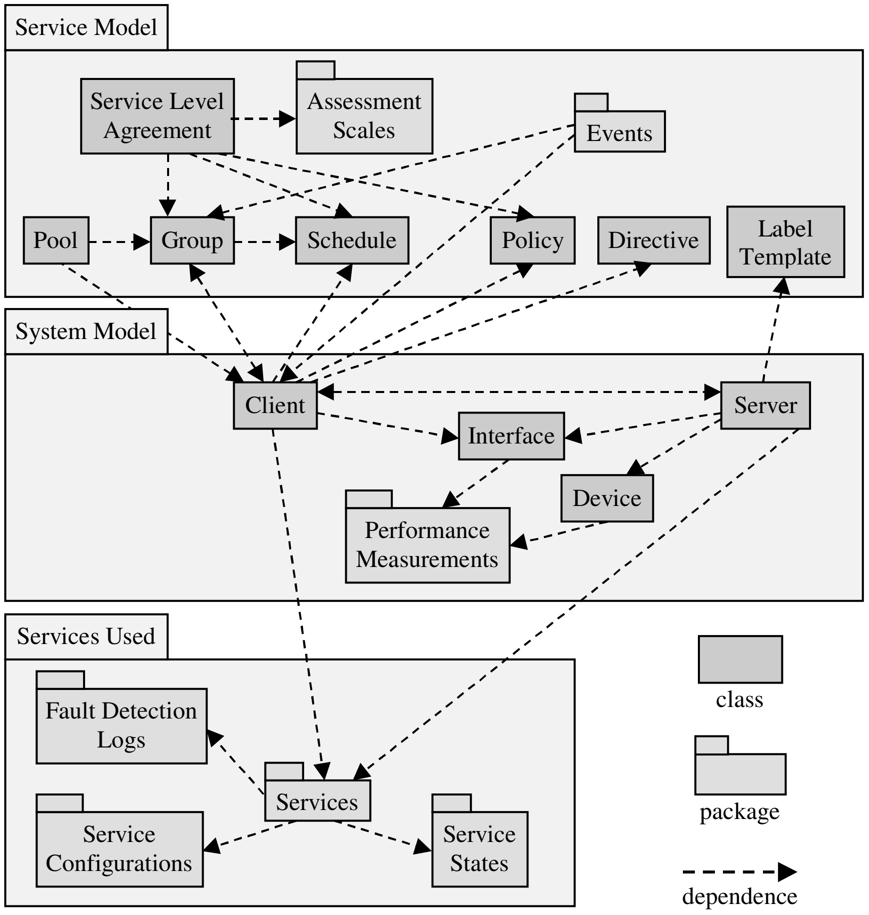

Fig. 1. UML package diagram of the model

The generation of the model for a given backup service is supported by a modeler tool and performed in four steps. At first the tool evaluates the backup system's configuration file. It generates and initializes most of the required object instances automatically. Secondly, the services used and the underlying resources are introduced. Here, the tool supports the guided interactive refinement of the model. The administrator can browse on the existing model and activate dialogs which create model objects and set attribute values and links. Thirdly, some *Services* of the *Services Used* package need detailed manual configuration. The SNMP-service for instance needs a port-number and a community-string to be set. Finally, some parameters of the management system may be adjusted manually (e.g., length of intervals for taking measurements, communication-protocol and port-numbers to be used for manager-agent communication).

5. Application Production and Model Utilization

The backup service management system has a traditional basic component structure. It consists of a manager application and a series of agents. The agents provide access to the managed objects comprising the backup application components as well as the objects of the services used by the backup application (i.e., host operating system services, backup device, file systems, and network interfaces). Moreover, the management systems contains the modeler tool. It supports the generation and interactive refinement of the model. Additionally, it communicates with the management application and provides the man-machine interface of the management system. Fig. 2 outlines this structure. On the right it shows the backup system, on the left the management system. Interfaces between management system and managed system are the backup system's configuration file – which is read by the backup system as well as by the management system – and the management agents' accesses to the managed objects. The modeler tool derives the code of the manager and decides the basic configuration of agents. It installs the manager which for its part installs the agents. During runtime the manager forwards current configuration and performance values to the modeler which displays convenient status representations to the administrator on demand.

The system model is represented by a configuration of object instances and maintained by the modeler tool as outlined in the last section. This object model is the source of information for the derivation of the code of the management system. For that purpose, the modeler tool transforms the model into an extended model which contains the model-specific management data and functions and will later on be part of the manager. Moreover, each agent will contain a copy of a small part of the model. The creation of the extended model coincides with the last two interactive model refinement steps. It is mainly performed by subclass refinement. For certain classes of the model (e.g., *Group*, *Client*, *Device*) corresponding subclasses are defined which add management-relevant attributes and methods. Object instances of the original classes are replaced by instances of the subclasses in the extended model. Initial values of new attributes are partially derived from an exploration of the existing model (e.g., direct links to used services in client objects). Other values are subject of interaction with the administrator (e.g., port-numbers and protocols to be used). The new methods directly perform management functions (e.g., installation and initialization of agents, fault isolation and recovery). Thus, the main parts of the manager application are supplied by the extended model. Moreover, the application code only contains a fixed frame and definitions of basic auxiliary classes. During runtime, the manager application maintains the extended model and updates status and performance attributes of its objects.

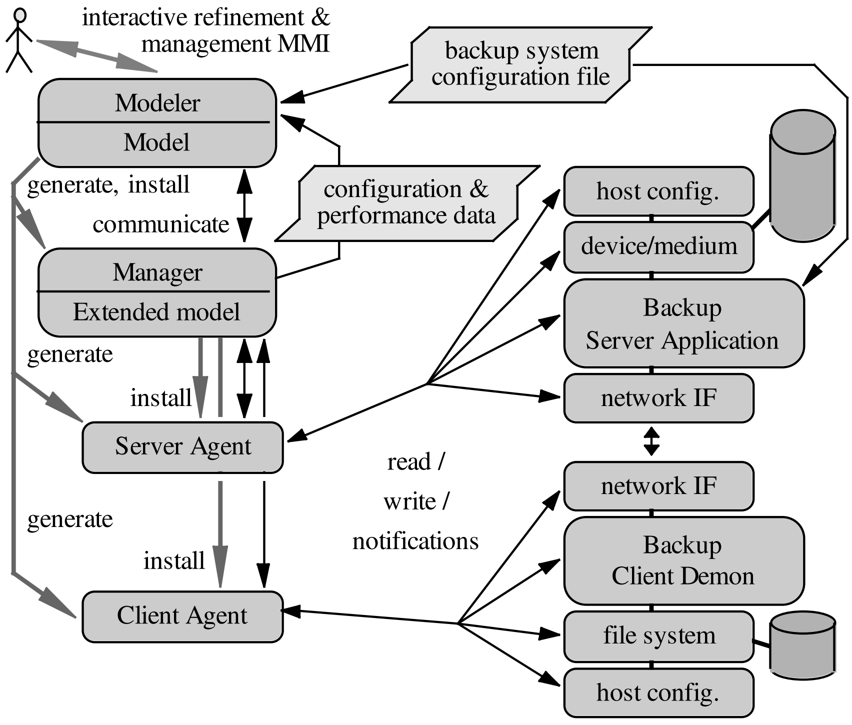

Fig. 2. Overview over management system and managed system

The subclass *ManagedClient* of the model class *Client* shall exemplify the refinement of the extended model. The attributes of *Client* mainly comprise lists of devices, save-groups and save-sets, configuration descriptions, and finally links to backup server, server interface, schedule, browse policy, retention policy, and save directives. The methods of *Client* perform reading and writing of data attributes, generation of graphical data representations, and creation of object copies. The subclass *ManagedClient* adds links to the used services (i.e., client-, network-, demon-, and SNMP-service), to the measurement scheduler, and to the agent frame. Moreover, data attributes represent the IP-address of the client and serve for the management of measurement timestamps. The methods manage and access the client's measurement scheduler, retrieve and restart the client's domain name server and backup demon, and finally reset measurement containers.

One important usage of the extended model is the service-oriented fault management. The handling of a negative save-group notification shall serve as corresponding example. Let the notification be created by the backup server application and received by the server agent of the management system. Then, the agent transforms the notification into a Java event and forwards it to the manager which is listed as event listener. The event mainly carries the name of the save-group, the number of clients, a list of faulty clients, timestamps, and a list of status data for each save-set (i.e., save-set name, client name, backup level used, duration of save, number of files and volumes, number of trials, list of files not saved, fault reason). Based on the extended model, the manager firstly resolves the save-set and client names. Thereafter the client reference serves as starting point and the fault reason serves as selector for an exploration of the connections of the client model object to the objects of the fault-relevant used services. For instance, the fault reason 'connection refused' can (besides of others) refer to a non-operational backup demon of the client. Therefore, the model exploration will reach the demon object. Its status is tested and in case of non-operation, the manager requests a demon restart from the agent of the corresponding client. Finally, the manager restarts the faulty backup subjobs. To avoid infinite recovery repetitions the performed recovery measures are recorded in the model and the number of retries is limited. Moreover, in connection with a formerly non-restartable backup demon the fault reason 'connection refused' can refer to a necessary reboot of a client. In this case the manager checks if it has the right to reboot this client and if so, initiates the reboot.

Another typical fault is the failing of the name/address-resolution due to non-operational or non-reachable domain name servers. In this case, the backup application is not able to connect to a client. In consequence of the fault event, the manager identifies the responsible domain name server and checks its status. Mostly the host is operational while the domain name server is down. Then the manager requests a restart of the domain name server from the corresponding domain name server management agent.

6. Implementation and Operation

The backup management system is implemented on the basis of the Java Dynamic Management Kit J-DMK [Jdm98] which applies the Java Bean component model [Bea98] in order to support flexible agents. Agent data structures and function implementations can be implemented by explicit and lately bindable code components, the so-called Beans. During runtime, agents can pull necessary Beans from a server or managers can push Beans to agents. The Beans act as runtime plugins and dynamically extend the agent's functionality. There are Beans implementing supporting services (like Core Management, Repository, SNMP-Communication). Moreover, so-called M-Beans implement management functions and the local access to managed objects (e.g., Backup Client Bean, Operating System Bean, Network Interface Bean).

Fig. 3 gives an overview over the J-DMK-based implementation. After the interactive definition of the model, the modeler produces serialized external forms of the manager and of the necessary agent Beans. After installation and start of the manager, it creates the necessary agents (one for each host) and establishes communication connections to them. Finally, it pushes to each agent the corresponding set of Beans.

The Bean set depends on the type of the host (backup client or server, type of operating system and network connection).

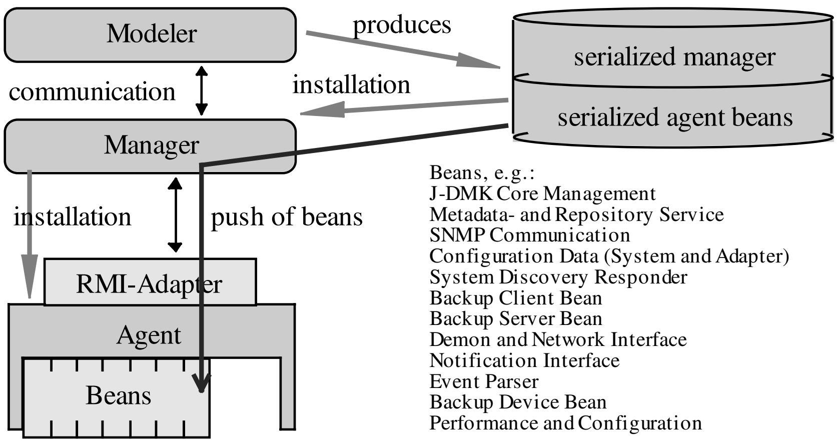

Fig. 3. J-DMK-based implementation

The whole management system – consisting of modeler, manager and agents – is implemented by approx. 400 Java classes (additionally, there are approx. 80 BeanInfo classes). The summed up length of the class files results in approx. 1.4 MB. The serialized form of the model needs about 20 KB per client host. The model-dependent parts of the manager need about 8 KB per client. The supply of model information to an agent has only to transfer about 9KB of serialized model substructures. The total byte code length of the components is: modeler approx. 1.4 MB, manager approx. 800 KB, server agent approx. 550 KB, client agent approx. 450 KB.

The management system operates in a local computer network which consists of approx. 120 backup client computers with various operating systems (i.e., Solaris, SunOS, HP-UX, Sinix, SCO, AIX, Windows NT). Backups are performed by one backup server which meanwhile is connected with an 500 GB DLT drive. Each night about 30 MB backup data are copied where full backups are achieved for approx. 6 clients (approx. 12 GB). Before the introduction of the backup management system mainly following faults caused unsuccessful subjobs:

- Failing name/address-resolution due to unavailable domain name servers (approx. 5 per month),
- Access conflicts to Network File System volumes due to incorrect mounts (approx. 10 per month, owing to a recent update of the backup application software now also most of these exceptions can successfully be handled by the backup application on its own),
- Backup demons on client hosts are unreachable or not operational (approx. 2 per month),
- Total client breakdowns (approx. 1 per month).

Now, the described backup service management system supports recovery for nearly all of the occurring faults resulting in a rate of unsuccessful backup subjobs of only approx. 1.5 per month.

7. Conclusion

We proposed a combination of service-oriented and model-based management which complies with two essential requirements of improved management systems, abstraction and automation, both enhancing the productivity of administrators. Service-orientation supports the identification and definition of abstract management objectives. Modeling supports the technology-independent consideration and description of application-internal dependencies thus supplying the information base for automated management functions. Both abstractions contribute to easy understanding and help administrators to deal efficiently with extended working domains as they result from modern integrated network, system, and application management tasks.

References

[Bea98] Sun Microsystems: Java Beans Specification; Sun Microsystems Inc., Palo Alto, 1998, available via http://java.sun.com/beans/docs/spec.html

[Cim98] Desktop Management Taskforce: Common Information Model - Specification 2.0; Desktop Management Taskforce Inc. DMTF, 1998, available via http://www.dmtf.org/spec/

[Eji96] M. Ejiri, S. Goyal (eds.): Proc. of the IEEE/IFIP Int. Symposium on Network Operations and Management NOMS'96, IEEE, 1996

[Enj96] A. Enjou, M. Tomobe: The Software Synthesis of Network Management Systems; in [Eji96], pg. 414-433, 1996

[Ere92] C. Batini, S. Ceri, S. Navathe: Conceptual Database Design - An Entity-Relationship Approach; Benjamin, Cummings, 1992

[Fer97] G. Fernandez, J. Derrick; Formal Specification and Testing of a Management Architecture; in [Laz97], pg. 473-484, 1997

[Gdm92] ISO / ITU-CCITT: Information Technology - Open Systems Interconnection - Structure of Management Information: Guidelines for the Definition of Managed Objects (GDMO); ISO-IEC IS 10165-4 also ITU/CCITT Recommendation X.722, 1992

[Gra95] G. Graw, V. Gruhn: Process Management in-the-Many; in: W. Schäfer (ed.) Software Process Technology, Proc. of the 4th European Software Process Modeling Workshop, Springer-Verlag, LNCS 913, pg. 163-178, 1995

[Hal96] J. Hall (ed.): Management of Telecommunication Systems and Services; LNCS 1116, Springer-Verlag, Berlin, 1996

[Haw98] M. Haworth: Service Management and Availability Planning for Data Backup and Recovery; HP Open View Service Management Solutions, White paper, Hewlett-Packard Company, Palo Alto, 1998

[Hei96] K. Heiler, R. Wies: Policy Driven Configuration Management of Network Devices; in [Eji96], pg. 674-689, 1996

[Jdm98] Sun Microsystems: Java Dynamic Management Kit; Sun Microsystems Inc., Palo Alto, 1998, available via http://www.sun.com/software/java-dynamic/

[Kae97] S. Kätker, M. Paterok: Fault Isolation and Event Correlation for Integrated Fault Management; in [Laz97], pg. 583-596, 1997

[Laz97] A. Lazar et al. (eds.): Integrated Network Management V, Proc. 5th IFIP/IEEE Int. Symposium on Integrated Network Management, Chapman & Hall, London, 1997

[Leg97] Legato NetWorker: The Key to High-Performance, Scaleable Storage Management; White paper, Legato Inc., Palo Alto, 1997, available via ftp://www.legato.com/legato/marcom/pdf/W008.html

[Lun96] A. Lundqvist, T. Grönberg: Network management simulators; in [Eji96], pg. 552-562, 1996

[Lup97] E. Lupu, M. Sloman: Conflict Analysis for Management Policies; in [Laz97], pg. 430-443, 1997

[Mcb98] D. McBride: Successful Deployment of IT Service Management in the Distributed Enterprise; White paper, Hewlett-Packard Company, Palo Alto, 1998

[Mib91] M. Rose, K. McCloghrie: Concise MIB Definitions; Internet Request for Comments RFC 1212, (1991)

[Ohs97] D. Ohsie, A. Maier, S. Kliger, S. Yemini: Event Modeling with the MODEL Language; in [Laz97], pg. 625-637, 1997

[Rod97] G. Rodosek, T. Kaiser: Determining the Availability of Distributed Applications; in [Laz97], pg. 207-218, 1997

[Sab97] M. Sabin, R. Russel, E. Freuder: Generating Diagnostic Tools for Network Fault Management; in [Laz97], pg. 700-711, 1997

[Tur93] K. Turner (ed.): Using Formal Description Techniques - An Introduction to Estelle, Lotos and SDL; John Wiley, New York, 1993

[Uml97] G. Booch, J. Rumbaugh, I. Jacobson: The Unified Modeling Language User Guide; Addison-Wesley, Reading, 1997

[War94] B. Warboys: Reflections on the Relationships Between BPR and Software Process Modeling; in P. Loucopoulos (ed.), Proc. 13th Int. Conf. on the Entity-Relationship Approach, Springer-Verlag, LNCS 881, pg. 1-9, 1994

[Wie94] R. Wies: Policies in Network and Systems Management - Formal Definition and Architecture; Journal of Network and Systems Management, 2,1(1994)63-83

[Wie95] R. Wies: Using a Classification of Management Policies for Policy Specification and Policy Transformation; in Proc. of the 4th IFIP/IEEE Int. Symposium on Integrated Network Management, Santa Barbara, 1995