A MANAGEMENT-AWARE SOFTWARE DEVELOPMENT PROCESS USING DESIGN PATTERNS

Oliver Mehl, Mike Becker, Andreas Köppel, Partho Paul, Daniel Zimmermann and Sebastian Abeck
{oliver.mehl, mike.becker, andreas.koeppel, partho.paul, daniel.zimmermann, sebastian.abeck}@cooperation-management.de
Cooperation & Management
Institute of Telematics, University of Karlsruhe
Zirkel 2, D-76128 Karlsruhe, Germany

Abstract: The provision of quality-assured IT services through a service provider requires that all IT components involved in these services can be managed in an efficient and effective way. The necessary management infrastructure must be adapted to these IT components and must be standard-based to allow its integration into an overall management environment. The broad spectrum of applications differing in functionality and architecture together with the need for a deep correlation of the management infrastructure with the internal structure and processes of an application make it difficult to use pre-defined application management solutions off-the-shelf.
The development process described in this paper addresses the problem by integrating the development of the management infrastructure into the software development process. The integration assures that the management infrastructure is adapted to the application to be managed. The infrastructure, including the management model as its core component, is based on the Common Information Model (CIM) standard. To support the development of the management model, a management design pattern catalog is introduced, that provides CIM-based patterns for the definition of standardized management models. The use of this catalog is demonstrated by an extension to the management model for the distributed Enterprise Resource Planning System R/3 of SAP AG.

Key words: application management, software development process, management infrastructure, management model, design pattern, CIM

1. INTRODUCTION

Modern software systems have to be able to provide IT services to customers in a quality-assured way. Therefore, every component that forms an application - including network-, system- and application components - must provide a management infrastructure including management models and corresponding instrumentation mechanisms.

Due to the broad structural variety and individual functionality of application architectures, the process to create an adequate and sound management infrastructure is a complex task. Additionally, in contrast to network or systems management, where management models can be built on dedicated base models, in most cases application management models must be built from scratch. An in-depth look into the application and its internal processes is necessary to ensure that the generated model reflects the application system appropriately. This puts the task of creating such a management model into the hands of the software developer. Being the person most familiar with the application, he is predestined to deal with the provisioning of the necessary management infrastructure.

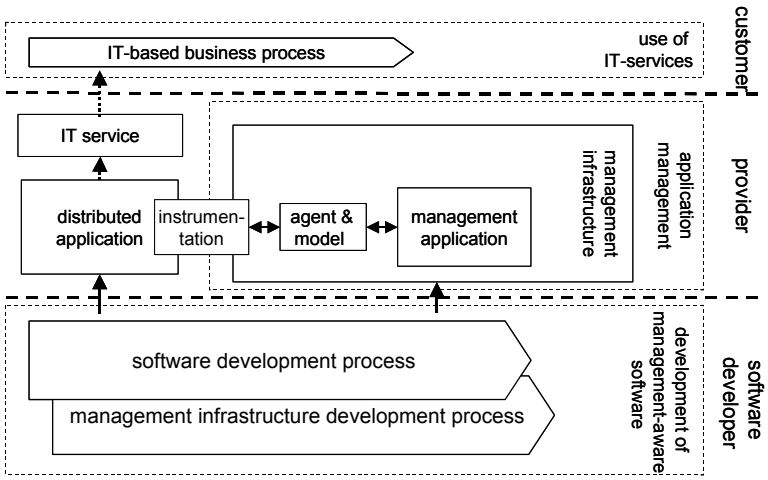

Figure 1. Development of management-aware software

To reduce the complexity of this task, an integrated process for the development of management-aware software systems as depicted in Figure 1 is necessary. A tight integration of the software and management infrastructure development processes avoids the problems of a too late alignment and harmonization of the application and its management infrastructure. To handle the additional complexity of this approach an integrated approach must provide guidelines and tools for application developers to ensure that management aspects are considered in all phases of the process.

Within this context the paper focuses on the development of the management model as part of the management infrastructure and demonstrates how the use of a design pattern catalog for management models supports the process. The design

patterns presented describe recurring parts of management models for application systems, modeled on a level of abstraction suited for management needs using a standardized management modeling language to ensure reusability and portability.

2. STATE OF THE ART

2.1 Management of applications

The complexity of application management [1] is not only driven by its various functional aspects but also by the broad spectrum of applications as managed objects. An important requirement for a successful management of applications is an adequate information model, which describes the applications' functional components and relations in the context of the business processes. Regarding the way to develop these models two basic approaches can be identified.

The most frequently used approach at present is a disjoint development of the software system and the corresponding management model. In this case, the management model and instrumentation code [2] need to be developed and integrated after the implementation of the application. If integration is impossible, the management must be based on information already available from the software, complemented by information gathered from an outside view of the application (e.g. system management information). This approach bears the risk of significant differences between the representation of the application in the management model and the actual software structure. The quality of the management achieved in this approach depends heavily on the degree to that additional instrumentation code can be integrated into the existing application system.

The second approach is to integrate the development of the management model into the software development process. This approach reduces the danger of inconsistent views on the application through a tight coordination between the development processes of both models. Due to their different aims and viewpoints a total match of the models is unlikely. Nevertheless, an integrated approach usually results in a well-adapted management view on the managed application that directly influences the quality of the overall management solution. The management-aware development process presented in this paper follows this integrated approach as described in detail in section 3.

2.2 Management infrastructure

In today's management scenarios required information for managing an application is often provided trough a specific management infrastructure in a non-standard way. To be integrated into a broader context such as service management management information must be accessible not only by application-specific management tools, that are tightly coupled to the managed application, but also by external management applications or platform such as Tivoli [3], BMC [4] or HP Openview [5] to enable correlation and aggregation of the management information.

A standard for modeling management information is the Common Information Model (CIM) [6] specified by the Distributed Management Task Force (DMTF). In contrast to management information languages such as SMI-GDMO (Structure of Management Information, Guidelines for the Definition of Managed Objects) [7], DMI-MIF (Desktop Management Interface, Management Information Format) [8] or SNMP-SMI (Simple Network Management Protocol, Structure of Management Information) [9] that are devoted to particular management domains CIM is an implementation and architecture-independent modeling language that allows to describe overall management information in a networked / enterprise environment [10]. It is flexible and extensible and therefore provides a wide range of applications. The rules for building CIM-compliant management models are defined in the CIM meta schema [6]. A set of basic elements to model management information is provided through the CIM base schema. CIM makes extensive use of the concepts of classes and instances known from object-oriented modeling and uses UML class diagrams to describe the models. This fact supports the idea to use CIM in an integrated development process in which the software developer must carry out the development of the management model as he is usually already familiar with these concepts and the notation.

The CIM compliant management infrastructure can be used to guarantee standardized access to an application's management information and functionality. To provide a manageable application that supports this approach, a developer has to provide the CIM-based management model, additional instrumentation code as well as a CIM provider for the application that can be registered with the CIM object manager (CIMOM) to make the CIM model, management data and functionality available to management applications.

3. AN INTEGRATED MANAGEMENT DEVELOPMENT PROCESS

The idea to integrate the management development process into existing software development processes assumes the identification of a generic core process. This process can then be extended to incorporate management aspects taking into account the integrated development of a standard-based management infrastructure for a management-aware software system. The integrated process can then be mapped to the original development processes to make them management-aware. Obviously the integrated development approach increases the overall complexity of the software development process that must be handled by the software developers. Even though a separate development team that is specialized in management details might transparently develop the complete management infrastructure, an intense synchronization between both processes is inevitable and results in additional costs. On the other hand the integrated process avoids the costly and error-prone process of adding management components after the implementation of the core functionality of a software system. The consideration of

management aspects right form the start of the application development improves the application's manageability in terms of the overall quality of its services.

When taking a closer look at established software development models such as the waterfall model, the V model or the spiral model [11] all of them share three main phases: a system analysis phase, a software design phase and an implementation and test phase. The integrated management development process depicted in Figure 2 is based on a software development process (SWD process) that consisting of these three shared phases. The management infrastructure development process (MID process) is separated into corresponding phases. The integration of both processes is done for each phase separately to ensure a consistent completion of each phase and by that to ease the integration of each of the phases into existing development processes.

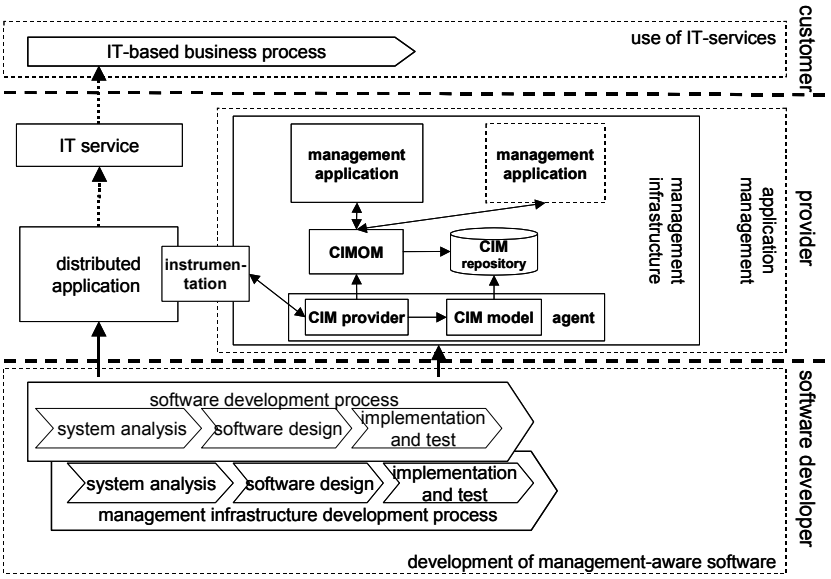

Figure 2. An integrated, CIM-based development process for management-aware software

To ensure a standardized access to the management information and functionality of the application the management infrastructure developed in this processes is supposed to be CIM-compliant. It deals with the development of the management model, the provider, the instrumentation code as well as the management application.

The following sections describe the first two phases of this development process. The last phase is left out as it concerns only implementation details.

3.1 The system analysis phase

The main focus of the system analysis phase is to determine and describe the requirements for an application system defined by the end user or customer. The requirements are documented in a product concept catalog and are later transferred into a customer requirement specification. Both documents describe the scope of the services of the developed application. In both cases the specification of the management requirements is realized after the requirements for the software have been defined as most management aspects are linked to or based on details of the software requirements analysis. To support the definition of management requirements a management catalog is used. The catalog provides a very high level classification of generic management information organized according to the five functional areas FCAPS of the OSI management [12]. It can be used in interviews to capture and define the customer's management requirements by instantiation and adaptation of the generic management aspects to the developed product.

After the requirements are defined, a product model is specified. Based on this model prototypes of the graphical user interfaces (GUI) are constructed that provide both, the software engineer and the customer, with a first vision of the application. In addition, a first version of a user manual is produced that is completed during the other two phases of the development process. If the development of a management application is necessary, the GUI for this application is specified in this phase following the same process but on a more general level. At first the management data provided by the management application is described in a broad outline through the definition of a first set of relevant management information and access functions for the application. This process is also supported by the management catalog that allows the definition of this data based on input from the requirement analysis. Additionally control mechanisms for the application are specified. Management data and management functions are then implemented in the management GUI.

The system analysis phase ends with the review of all the documents and specifications that have been created during this phase to guarantee that the specifications meet the requirements. In case of inconsistencies, changes or extensions to the existing specifications must be carried out.

3.2 The software design phase

The software design phase depicted in Figure 3 deals with the transformation of the specified requirements into a software architecture. During this process constraints and limiting conditions are refined and extended based on the outcomes of the system analysis phase. In addition, operation conditions are defined.

The software architecture describes the structure of the software system by defining the system components and their relationships. A system component is a closed part of a software system dealing with a specific (functional) aspect of the application. Depending on the level of detail the components are organized into layers or tiers whereas logical layers are mapped onto physical layers. The mapping of the components can be done following predefined schemes.

After the system components and the architecture have been defined, the components are specified in detail. This includes the definition and documentation of their interfaces as well as their internal behavior in a formal or semi-formal way. Tools supporting this process are e.g. UML CASE tools to create static (class diagrams) or dynamic (activity or sequence diagrams) diagrams of the components. Due to its complexity, the process of defining the software architecture and its components is carried out iteratively, refining the results of the preceding phase. If necessary, results must also be fed back into prior phases of the design process to change or extend earlier specifications or constraints.

The design of the management model can be initiated after the system components and the overall architecture have been defined. The management model can include various kinds of management information from the different management areas depending on the focus of the management. Constraints and decisions taken in the design of the application architecture influence the development of the management model. They must be taken into account to ensure a consistent and sound management view of the application. To ease the development of the management model, a management design pattern catalog is used. The purpose of this catalog, its structure and usage is described in section 4 in detail.

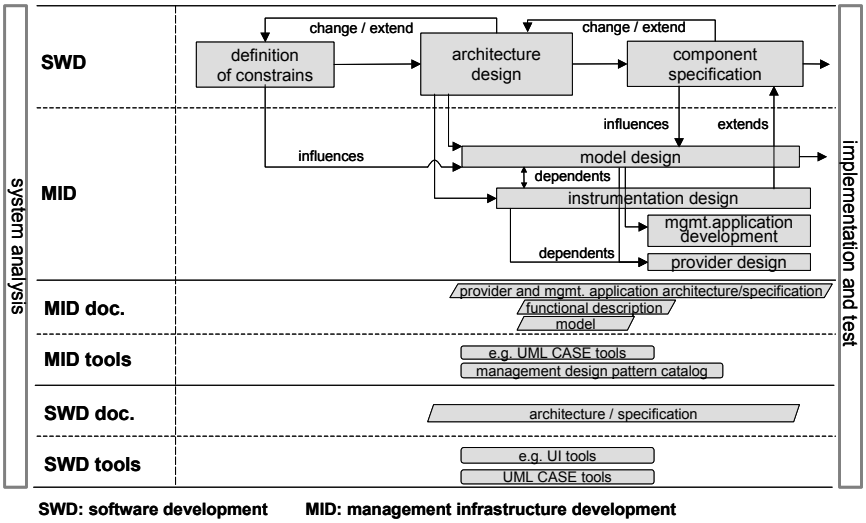

Figure 3. Management-aware software design phase

The management information and functionality is provided by the application components through the instrumentation. Therefore the instrumentation requirements extend the component specification as they define additional functions and parameters that need to be integrated into the components. The instrumentation also influences the design of the management model: only aspects that can be covered by the instrumentation can be instantiated in the management model.

After the management model and instrumentation design become more stable, the design of the management provider and, if necessary, the management

application development is initiated. The provider is responsible to transfer the management data received from the instrumentation to the management application (via the CIMOM) and must therefore implement the management model. The management application is designed and refined according to the requirements and the initial GUI design defined in the system analysis phase. Based on the information represented in the management model, details about management data presentation and possible data manipulation operations are specified.

4. A MANAGEMENT DESIGN PATTERN CATALOG FOR THE SOFTWARE DESIGN PHASE

The component- or object-oriented way to design the CIM-compliant management model in the integrated development process fosters the reuse of design models in the form of patterns that solve recurring design problems. In software engineering design pattern consist of a combination of components or classes, their interfaces and relationships that reflect important aspects of a solution to a specific problem. The reuse of these patterns is only possible if they provide a sufficient level of abstraction from both the underlying problem and the solution. While software design patterns usually handle details of the system design, patterns used to build management models operate on a higher level of abstraction relying on component- and function-oriented view of an application system. Following this approach a management design pattern catalog can be used to assist in the development of management models. Using such a catalog provides several benefits: The developer of the management model gets a set of building blocks that allow an efficient implementation of the model. The patterns also support the development of the application system instrumentation as they specify operations and attributes that must be provided by the application to enable its management. Both aspects reduce the effort to develop an appropriate management infrastructure for an application as only additional application-specific characteristic that are not covered by the patterns must be added to the model. Additionally, the use of management design patterns stimulates the development of comparable management models. By basing the representation of application system components in the management model on well-defined building blocks the interoperability with other models providing similar attributes and inheritance hierarchies is supported.

4.1 Management design pattern catalog

In contrast to application software design the management model design focuses on the functional units and services of an application on a reasonable level for management tasks. When analyzing the functional aspects of the application various indicators help the developer to discover such entities. Examples are components supporting asynchronous data processing or boundaries between software components that are embedded in separate executables or communicate remotely.

The management design pattern catalog supports the developer in identifying such entities as well as in the selection of the parts of these entity that are relevant to be include in the management model. The resulting requirements for the instrumentation of the application system can then be fed back into the application design process. The level of abstraction used for the patterns is located in between the rather abstract concept of architecture styles (see [13]) and the fine-grained software design patterns used in the software design process. The patterns focus on the description of logical processing elements such as services and sub-systems. The patterns are described following the approach used in [14] to ease their selection and use by software developers. Therefore each pattern in the catalog is described by its name, a short overview as well as examples for its use. Synonyms and related terms are mentioned to ease navigation and search for an appropriate pattern. While this information is intended to support the selection of a pattern from the catalog, each pattern is also described in detail and visualized by a CIM model. Preconditions and limitations for the use of the pattern are mentioned.

The initial set of design patterns for the management design pattern catalog were identified during the design of a management infrastructure for the Enterprise Resource Planning System (ERP System) R/3 of SAP AG. A strong effort during the analysis was put on keeping the identified patterns independent from the specific characteristics of the SAP software to guarantee their reusability and to define them on a suitable level of abstraction to keep them applicable for other scenarios.

Two high-level patterns were identified as starting points for further analysis:

- The basic structure of an application system as a class model is presented in the *distributed application system* pattern.
- Systems featuring logical segments with private data and configuration sections may be modeled using the *logical system* pattern. A web server providing services for different virtual sites on the same hard- and software platform can be taken as an example for such a system.

Design patterns for single services that were identified in the functional analysis include:

- An *application service* represents a service that provides a set of related functions in an application system.
- A *distributed service* is provided by services of different component systems. A load balancing service can be taken as an example for this pattern.
- A *specialized service* represents a service implemented from specialized components or component systems within a distributed application system, playing a vital role as a single point of failure for the system.
- A *queued service* uses data queues or asynchronous request processing to provide its service. An example is a request processing service of a print server.

In addition, the following patterns can be used to describe more general component relationships.

- The pattern *client-server relationship* represents the relationships between two components that are part of a service provide/service user relationship. The

communication relation ship between business tier components and a database can be described using this pattern.
- To create a model describing the properties of an external component that is used by an application system the pattern *external view* may be used.

4.2 Pattern examples in detail

In the following sections the design patterns *distributed service* and *specialized service* are described in detail using the catalog structure introduced in section 4.1.

4.2.1 Application Service

Overview: An *application service* combines a set of functions of a technical service. All processing elements of an application can be mapped to such a service. Criteria for this kind of services are a fixed interface and permanent service availability.
Example: A spool service for print jobs or a web server module such as a SSL module for communication encryption can be modeled as application service.
Related terms: functional group, processing element, module

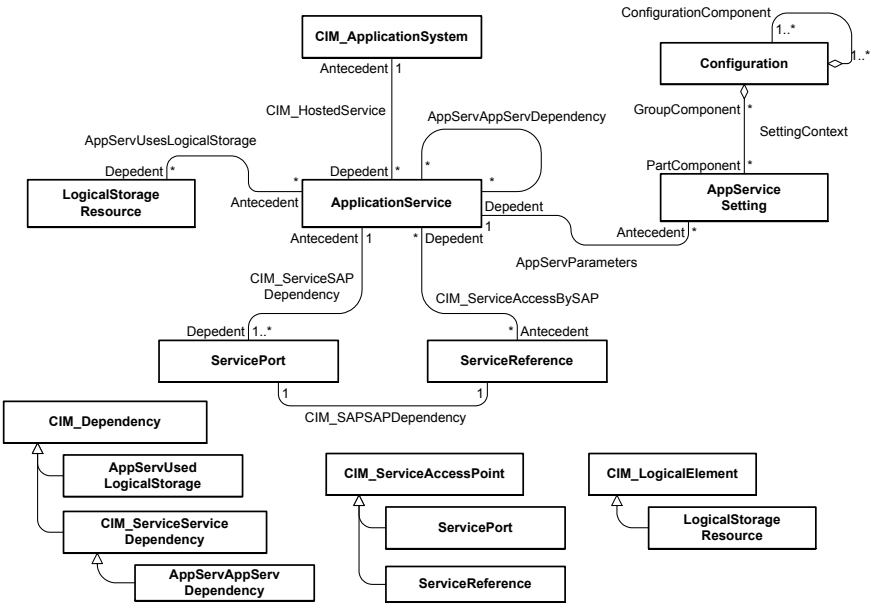

Figure 4. CIM model of the management design pattern *application service*

Motivation: The pattern allows to model the management aspects of any relevant components of an application system from a functional point of view.
modeling details: The *ApplicationService* class is linked to the *CIM_Application-System* class that implements the service functionality. It can use other application

services to implement its functions or provide its own functionality to other services. This is done by the use of a *ServiceReference* (to use functions of other services) or a *ServicePort* (to provide functionality) that are derived from the class *CIM_Service-AccessPoint*. The use of data storage systems is modeled through the class *Logical-StorageResource*. Configuration options are mapped to the *AppServiceSetting* class derived from *CIM_Configuration* that represents a part of the overall application settings.
Preconditions and limitations: The key problem to use this pattern is the identification of the relevant elements of an application that should be modeled as application services. Therefore an adequate level of abstraction of the software component model must be chosen before this pattern can be used.
Visualization: See Figure 4

4.2.2 Specialized Service

Overview: The pattern *specialized service* represents a service that is implemented only by specific components and is of central importance for the overall system.
Example: The payment transaction component of an e-commerce application can be seen as a specialized service. It is the single interface of the application providing access to the bank to carry out financial transactions and it is used by all other components of the e-commerce application requiring this functionality.
Related terms: Dispatcher, master, single point of failure

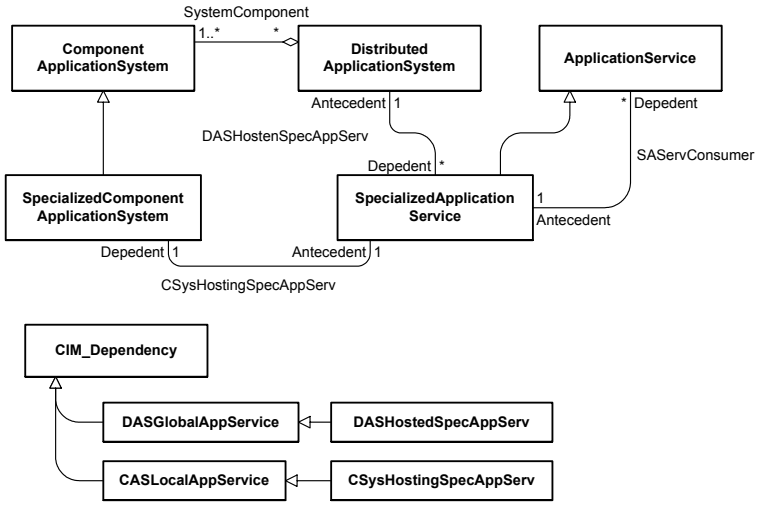

Figure 5. CIM model of the management design pattern *specialized service*

Motivation: Specialized services are of special interest from a management point of view regarding fault or performance management. A failure on their part has major implications on the overall system availability. Their performance influences the overall productivity of the system.

Modeling details: A specialized application service is modeled by the *Specialized-ApplicationService* class derived from the *ApplicationService* class. This class is associated with a component system *SpecializedComponentApplicationSystem* that provides the service. The specialization of this system does not necessarily involve special software components but can also be achieved through specific configuration options. The service is used by other application services indicated by the *SAServConsumer* association to the *ApplicationService* class.
Preconditions and limitations: The overall application system must be built from different component systems. The services provided by the application system must be provided through component-based services. The relationships between these services must be known to allow the identification of specialized services. The pattern does not represent the functional aspects of the relationships.
Visualization: See Figure 5

5. USE OF MANAGEMENT DESIGN PATTERNS IN THE CONTEXT OF THE ERP SYSTEM SAP R/3

The overall architecture of the ERP system SAP R/3 follows a distributed three-tier architecture. In this scenario the database tier is a centralized external relational database providing its services to the business tier. In the business tier so called work processes use this central database to provide their functionality in the context of various business processes. The work processes are grouped to application servers that are located on different physical systems. Each application server consists of exactly one dispatcher process and an optional gateway service to handle the distribution of requests between the local work processes and the communication between the applications servers of a single SAP system.

The Internet Communication Manager (ICMan) component enables an application server to communicate using IP-based protocols such as HTTP or SMTP. This is achieved by extending the application server by a separate ICMan process that can be addressed by the work processes through the dispatcher process.

In the following the use of the management pattern catalog is demonstrated in the development of a management model for the ICMan.

5.1 Management model for the ICMan component

Based on the functional and architectural aspects of the ICMan component the management pattern *specialized service* was identified as best choice to develop the management model. The service provided by the ICMan can be characterized as specialized service, because its service is provided by not more than one component instance but it is accessed by several other components to provide their services. Even though the ICMan component can't be seen as central in the overall application context it is central in the context of its associated application server.

Figure 6 depicts the CIM model of the ICMan component as it was integrated into the management model of the SAP base system. According to the design pattern

specialized service the ICMan component was modeled as a CIM class *SAP_BCInternetCommMgrService* derived from the abstract class *SAP_Application-Service* that corresponds to the class *ApplicationService* in the design pattern. The association *CsysHostingSpecialService* is instantiated as *SAP_BCKernelICMan-Implementation* and is linked to the *SpecializedComponentApplicationSystem* that implements the component system including the ICMan, which in this case is represented by *SAP_BCKernel*. As the service of the ICMan is used by the work processes of the application server the dependency *SAServConsumer* is modeled by the association *SAP_BCICManProvidesServiceToWP* between the *SAP_BCInternet-CommMgrService* class and the *SAP_BCWorkProcess* that correspond to the *ApplicationService* class in the pattern.

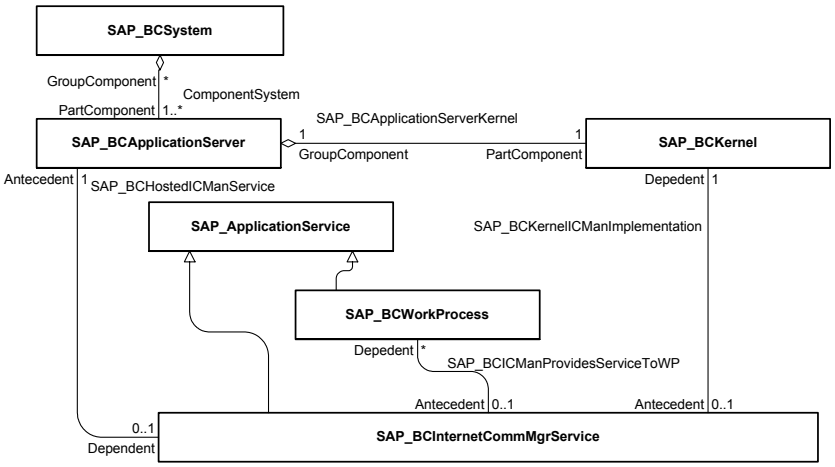

Figure 6. CIM model of the ICMan component

6. CONCLUSION

The management-aware development process depicted in this paper presents an approach for the integrated development of applications and their management infrastructure by the inclusion of management aspects in all phases of the software development. It fosters the alignment and integration of the management infrastructure, weaving the management model with the application system in a standardized way. The design pattern catalog presented supports application developers to handle the additional complexity of the integrated development process. It provides as set of patterns to ease the construction of CIM-based management models as an important building block of the management infrastructure. The use of the pattern catalog was demonstrated in the development of the management model of a central component of an ERP system.

The integrated development process is currently being evaluated in the development of different management-aware components. The results are fed back into the ongoing improvement of the process and the supporting tool set. Actual activities focus on the specification of an extended schema for a product concept catalog and a customer requirement specification used in the system analysis phases as well as customized methods for using them efficiently. As one of the supporting tools the design pattern catalog is also continuously evaluate, revised and extended by modeling further components of the SAP ERP system to improve and verify the reusability of the design patterns defined so far.

REFERENCES

[1] R. Sturm and W. Bumpus, "Foundations of Application Management": John Wiley & Sons, 1999.

[2] M. Katchabaw, S. Howard, H. Lutfiyya, A. Marshall, and M. Bauer, "Making Distributed Applications Manageable Through Instrumentation", presented at 2nd International Workshop on Software Engineering for Parallel and Distributed Systems (PDSE'97), 1997.

[3] IBM, "Tivoli® Software", http://www.tivoli.com.

[4] B. Software, "Enterprise Applications Management", http://www.bmc.com.

[5] H.-P. Company, "HP OpenView", http://www.openview.hp.com.

[6] DMTF, "Common Information Model (CIM) Specification, Version 2.2", http://www.dmtf.org/standards/cim_spec_v22/.

[7] ISO, "International Organization for Standardization - Information Technology - Open Systems Interconnection - Structure of Management Information: Guidelines for the Definition of Managed Objects, ISO 10165-4", 1992.

[8] DMTF, "DMTF, Desktop Management Interface (DMI) Standards", http://www.dmtf.org/standards/standard_dmi.php, 1998.

[9] IETF, "RFC1155 - Structure and Identification of Management Information for TCP/IP-based Internets , Internet Engineering Taskforce", 1990.

[10] W. Bumpus, "Common Information Model: implementing the object model for enterprise management". New York, N.Y: Wiley, 2000.

[11] G. Kotonya and I. Sommerville, "Requirements Enginerring": John Wiley & Sons, 2000.

[12] ISO, "International Organization for Standardization - Information Technology - Open Systems Interconnection - System Management: Object Management Function, ISO 10164-1", 1993.

[13] D. Garlan and M. Shaw, "An Introduction to Software Architecture", vol. Volume I: World Scientific Publishing Company, New Jersey, 1994.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: elements of reusable object-oriented software": Addison-Wesley Publishing Company, Reading, MA, 1995.