DESIGN AND IMPLEMENTATION ASPECTS OF MANAGEMENT SYSTEMS FOR APPLICATION SERVICE PROVIDING

Michael Höding
University of Applied Sciences Brandenburg, PF 2132, D-18737 Brandenburg, Germany

Keywords: Application Service Providing, Design Patterns, System Monitoring.

Abstract: This contribution discusses management software for Application Service providing (ASP). For the development of such software systems numerous specific requirements have to be considered. As examples we discuss aspects of heterogeneity. Because of this a flexible software engineering approach is necessary, covering design and implementation. For that we propose design patterns and component technology. The application of design patterns is demonstrated in examples.

1 INTRODUCTION

Application Service Providing (ASP) is a business model to support medium scaled enterprises with highly integrated and complex business software (Kern and Kreijger, 2001), (Tao, 2001), (Knolmayer, 2002). The Application Service Provider implements a technical and organisational infrastructure that guarantees a high degree of data security and system availability. Several customers share the use of the computing centre infrastructure (ref. fig. 1).

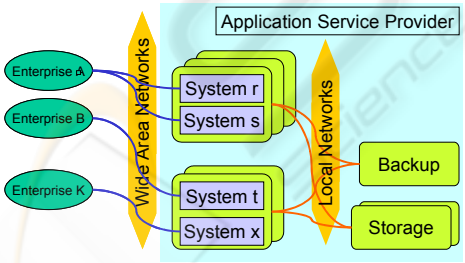

Figure 1: Customers and Components in ASP

The technical administration of a computing centre for ASP should be supported by an administration or monitoring software (Hoding and Faustmann, 2001). This software has to cover a wide range of hardware and software components. The design and implementation of such a system is driven by numerous requirements:

- Distribution: An ASP installation is, by default, a distributed system, containing several server computers and specific devices, e.g. storage or backup. Beside this, also non-IT-components have to be covered.
- Multi-tier-architecture: Generally software hosted with ASP is client/server-based. Mainly a 3-tier-architecture (user interface, application server, database server) is used.
- Flexibility of the system (during runtime): Due to the necessary degree of availability the ASP installation as well as the management system has to be enhanced during runtime. Therefore flexibility of the management system is a key requirement.
- Different kinds of heterogeneity exist despite the same software
- Usability needs results in the requirement of management from global view to in deep analysis by different representations (views)

Obviously one has to deal with hardware heterogeneity. Beside a number of server computers, different kinds of sharable hardware components, e.g. storage devices or backup devices have to be considered. Moreover infrastructural hardware like air condition enhances this aspect of heterogeneity which has to be controlled by the management software.

Höding M. (2005).
DESIGN AND IMPLEMENTATION ASPECTS OF MANAGEMENT SYSTEMS FOR APPLICATION SERVICE PROVIDING.
In *Proceedings of the Second International Conference on e-Business and Telecommunication Networks*, pages 372-375
DOI: 10.5220/0001419703720375

Despite of the fact, that sometimes an ASP offers only one (or a limited number of) standard software, heterogeneity can be found on different levels of system distribution. This is caused by different requirements of the customers, e.g. in workload. Figure 2shows the distribution of a classical 3-tier-architecture, consisting of user interface, application server and database server. Generally the user interface is running on a dedicated client computer. A common approach implements application server and database server on one server computer (1 at 1). Powerful servers can host more than one application system (many at 1). Otherwise systems with high workload can distribute many application servers and many database servers on dedicated computers (1 at many). Finally we have to point out, that these configurations are relevant for the same software. For example: “many at 1” and “1 at 1” are common installations of SAP R/3. Another example is web service providing which is offering variants from shared server, dedicated server up to server farms.

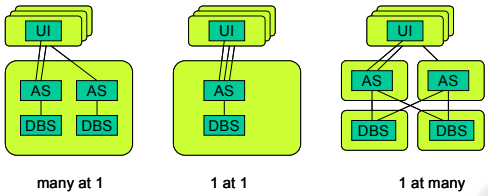

Figure 2: Distribution Variants of a 3-Tier-System

Based on a more detailed and systematic description of foundations and core requirements of a distributed administration system for ASP we discuss the application of object-oriented techniques for modelling and implementation of such a system. Therefore first a common approach defines an interface to non object-oriented monitor modules by encapsulation into a monitor object (ref. fig. 3). This is called a wrapper.

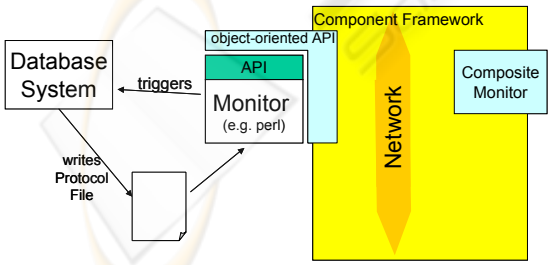

Figure 3: Monitoring and Wrapping

2 RELATED WORK

The design of a complex object-oriented system can be supported by the methodology of design patterns presented in (Gamma et. al, 1994). In that way, the requirements can be applied to a pattern catalogue to design suitable object models.

First we discuss the *Bridge* pattern (fig 4). The intention of the bridge is the design of an abstract interface to one or more concrete implementations. The loose coupling by a message based notification algorithm results in advances like flexible activation of new implementations during runtime or encapsulation of heterogeneity, i.e. for not object-oriented monitor kernels.

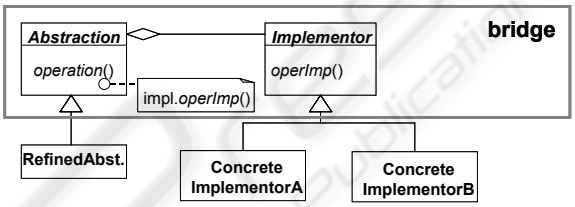

Figure 4: The Bridge Pattern (Gamma et. al, 1994)

Figure 5 shows the *Observer* pattern. Here an observed object (Subject) is loosely coupled with one or many observing objects (Observer). There the abstract objects Subject and Observer are specialised into ConcreteSubject and ConcreteObservers. New observers register themself with the subject method attach() during runtime. The subject notifies his registered observers when its internal state is changed.

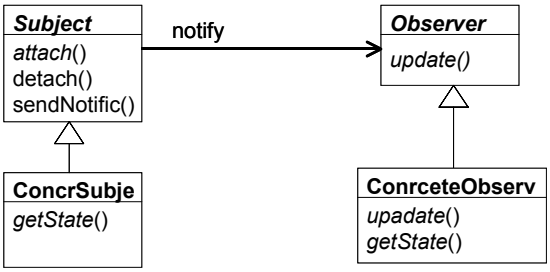

Figure 5: The Observer Pattern (Gamma et. al, 1994)

With respect to the fact of distribution we refer to patterns for distributed systems, e.g. as presented in (Brown et. al, 1999). Here specific design patterns for distributed systems support the aspect of communication in networked environments.

The implementation of a distributed administration software for ASP can be done by available systems e.g. Tivoli (Uelpenich, 1999) or PATROL (Boeheim, 1997). To avoid the administration overhead by available software

products a dedicated implementation for the given computing center architecture can be suitable. For such an implementation component frameworks e.g. Java Beans (Matena and Stearns, 2001) can cover heterogeneity on hardware and operating system level.

SOAP is often used for the communication between heterogeneous components (Apps, 2004). The automatic generation of communication interfaces is supported by different component frameworks, e.g. Java and .net. Hereby monitors, implemented in C#, can bee accessed by a Java-Class and vice versa.

3 USE BRIDGE PATTERN TO ENCAPSULATE SPECIFIC MONITORS

Often monitors for specific system components are based on non object-oriented techniques, e.g. parsing and filtering log files or accessing internal runtime information by operating system calls. For such concrete implementations the bridge supports an abstract interface as illustrated in the figure 6. Here we depict two concrete monitor implementations for accessing status logs (FileMonitor) and for accessing the process table of the operating system (ProcMonitor). Via the bridge and the abstract monitor class specific monitors, e.g. for the Oracle DB software can use the available implementations.

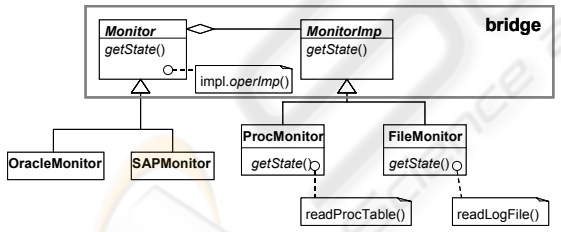

Figure 6: Bridge Pattern for Viewers and complex Monitor

However other patterns should be discussed to solve this problem. The adapter pattern as well as the wrapper pattern well-known from CORBA can support suitable solutions. The application of the *Model-View-Controler* pattern (Gamma et. al, 1994) offers two interesting aspects. First, every monitor can be designed according to MVC. Here the model encapsulates the specific request to the observed system, e.g. reading and matching a log file. The views support a set of representations in different granularity. In that way a basic monitor is directly usable for the administration staff. Second, the monitor functions primary as a model according to the MVC pattern. View and controller function is part of higher levels of the architecture. This simplifies the implementation (keep it small and simple) and enhanced the performance. The first idea we will illustrate in the next section.

4 USE OBSERVER PATTERN TO CONSTRUCT COMPLEX MONITORS

As depicted before a Monitor is a specific software module to control a component in the system, e.g. the hardware of a server computer or the database management software. This monitor is an observed subject. As examples we construct two types of observers:

- First for every specific monitor one (or many) viewer represents the state of a monitor by a graphical user interface. This is an application of the Model-View-Controller concept (Gamma et. al, 1994).
- Second a complex monitor aggregates simple or other complex monitors to support a more general view to the system.

Figure 7 shows the derivation of a Monitor from the abstract Subject class. Concrete observers, here ComplexMonitor and MonitorViewer, are derived from the abstract Observer.

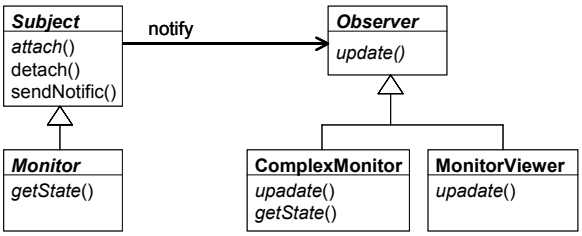

Figure 7: Use of Observer Pattern to construct complex Monitors

An instance of such implementation is sketched in figure 8. Here the operating system of one server computer, the database system Oracle, and two SAP application servers are monitored by OSMonitor, OracleMonitor, and two SAPMonitors. For a SAPMonitor an observing viewer shows the general state of the monitored systems using a traffic light as a metapher (Green means OK, Yellow means warning, Red means critical). For the OSMonitor a

viewer shows workload aspects by the means of a diagram. Beside the viewers a ComplexMonitor aggregates all simple monitors. In that way a flexible architecture supports different types of controlled system components. Moreover a general view could be supported constructing a viewer to the complex monitor.

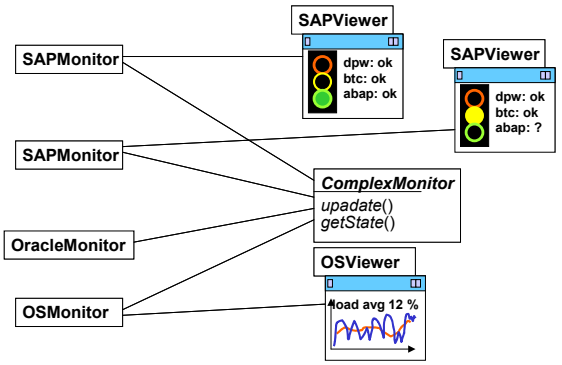

Figure 8: Monitor observed by complex Monitors and Viewers

5 CONCLUSION AND OUTLOOK

This contribution discusses the design of management systems for complex computing centre infrastructures, especially for ASP. For that we propose the application of design patterns. We sketched the application of design patterns by the means of two examples. In future work a more detailed study has to consider additional design patterns. I. e., the factory pattern should be a good solution to clone monitor objects or viewers for distribution. Finally we have to point out, that patterns help to meet the requirement of flexibility, also during runtime. For implementation issues a component framework should be used. Thereby the requirements distribution and heterogeneity on platform level can be fulfilled.

Further work is dealing with a classification of services and components which have to be observed. There are dimensions e.g. time, time span, volume, traffic, usage, weather or profile. The multi-dimensional space of these vectors is associated with so called system events, e.g. problem situations. To improve quality approaches like data-mining should be adopted.

REFERENCES

Apps. A.: *zetoc SOAP: A Web Services Interface for a Digital Library Resource* in Proc. Of Research and Advanced Technology for Digital Libraries, 8th European Conference, ECDL 2004, Bath, UK, Sept., 2004.

C. Boeheim *Patrol.* Sys Admin: The Journal for UNIX Systems Administrators, 6(2), p. 16-22, February 1997.

K. Brown, P. Eskelin, N. Pryce. *A Mini-Pattern Language for Distributed Component Design.* PLoP 1999 Conference. Pattern Languages of Programs, August 1999.

E. Gamma, R. Helm, R. Johnson, an J. Vlissides: *Design Patterns Elements of Reusable Object-Oriented Software*: Addison Wesley, 1994.

M. Höding, A. Faustmann: *Administration of data intensive Application-Hosting-Scenarios (In German).* Foundations of Database Systems 2001: 63-67

T. Kern, J. Kreijger: *An Exploration of the Application Service Provision Outsourcing Option.* HICSS 2001

Knolmayer, G., *Application Service Providing (ASP)*, in: Wirtschaftsinformatik 42 (2000) 5, S.443-446.

V. Matena, B. Stearns. *Applying Enterprise JavaBeans: Component-Based Development for the J2EE Platform.* Addison-Wesley, 2001.

D. C. Schmidt, P. Stephenson: *Experience Using Design Patterns to Evolve Communication Software Across Diverse OS Platforms.* ECOOP 1995: 399-423.

L. Tao: *Shifting Paradigms with the Application Service Provider Model*. IEEE Computer 34(10): 32-39 (2001)

S. Uelpenich: *Extending the Reach of Tivoli Distributed Monitoring - Creating a Custom Monitoring Collection.* The Managed View, 3(2), pp. 21-40, Springer 1999.