Software Architecture Patterns for System Administration Support

ROLAND BIJVANK , WIEBE WIERSEMA and CHRISTIAN KÖPPE, HU University of Applied Sciences, Utrecht, the Netherlands
{roland.bijvank,wiebe.wiersema,christian.koppe}@hu.nl

Many quality aspects of software systems are addressed in the existing literature on software architecture patterns. But the aspect of system administration seems to be a bit overlooked, even though it is as important as other aspects. In this work we start with mining the software architecture patterns that support the work of system administrators.

Categories and Subject Descriptors: D.2.10 [**Software Engineering**]: Design—*Design Patterns*

General Terms: Software Architecture, System Administration

Additional Key Words and Phrases: Software Architecture, System Administration

ACM Reference Format:

Bijvank, R., Wiersema, W., Köppe, C. 2013. Software Architecture Patterns for System Administration Support – L(October 2013), 7 pages.

1. INTRODUCTION

In the article *A plea from sysadmins to software vendors: 10 Do's and Don'ts* by Thomas Limoncelli, published in the Communications of the ACM magazine [Limoncelli 2011], system administrators collected a basic list of what software vendors should do and not do in order to make the life of the system administrators more easy. It seems that in certain points there is a high agreement between administrators what best practices should be. However, these practices often belong to another discipline, namely software architecture.

The focus of Software Architecture is often on realizing quality attributes, such as those described in ISO 25010: Functional suitability, Performance efficiency, Compatibility, Usability, Reliability, Security, Maintainability and Portability. Many patterns have been described, e.g. in the POSA book series [Buschmann et al. 1996], and their general applicability for realizing the qualities has been discussed [Harrison 2011]. There are also publications that focus on patterns for specific quality aspects, like patterns for fault tolerant systems [Hanmer 2007] or security patterns [Schumacher et al. 2005]. But there is one quality attribute where not much attention has been paid to: Portability and its sub-qualities Adaptability, Installability, and Replaceability. A number of concerns from system administrators are covered by the aforementioned attributes, but the mapping of the concerns on the attributes is not intuitive.

There have been several initiatives to describe patterns from the perspective of a system administrator, but these are mainly focused on infrastructure and middleware. Examples of these initiatives are:

—Daniel Jumelet: Open Infrastructure Architecture repository (OIAr)[1] - this site provides a wide variety of infrastructure patterns for several working areas: Client Realm, Middleware, Network, Security + Support, Server, Storage. Beside this repository also contains architecture & design guidelines in the form of construction models at various levels and from various angles. It is constructed by making use of one of the most important tools of OIAm: The Building Blocks

[1] http://www.infra-repository.org/oiar/index.php/Main_Page

Model. The Building Blocks Model is primarily a *decomposition* tool. That means that it is used to dissect infrastructure landscapes into logical dimensions and parts in order to enable structured and methodological modeling (*composition*).

—Gregor Hohpe and Bobby Woolf: Enterprise Integration Patterns[2] - this site provides a consistent vocabulary and visual notation to describe large-scale integration solutions across many implementation technologies. It also explores in detail the advantages and limitations of asynchronous messaging architectures.

Both approaches — software architecture patterns for realizing the above described quality attributes and patterns that support the work of system administrators — don't touch some important aspects of the intersection of software architecture and system administration. Therefore we want to introduce a set of patterns which bridges this gap, based on the needs of the system administrators.

The problems that are cited in the aforementioned article have been experienced within daily system administration practice.

With these patterns we want to give ideas to software architects and application developers on how to improve their applications from a system administration viewpoint.

2. THE PATTERNS

In this paper we present the first four software architecture patterns for system administration support:

—PROVIDE AN ADMINISTRATION API

—SINGLE FILE LOCATION

—USE BUILT-IN SYSTEM LOGGING

—CENTRALIZED IDENTITY MANAGEMENT

The patterns use a version of the Alexandrian pattern format, as described in [Alexander et al. 1977]. The first part of each pattern is a short description of the context, followed by three diamonds. In the second part, the problem (in bold) and the forces are described, followed by another three diamonds. The third part offers the solution (again in bold), consequences of the pattern application — which are part of the resulting context — and a discussion of possible implementations. In the final part of each pattern, shown in *italics*, we discuss related patterns and offer a rationale for the pattern based on literature.

[2] http://www.eaipatterns.com/

PROVIDE AN ADMINISTRATION API

The system has to provide an API (application programming interface) that allows the system administrator to automate system administration tasks. Examples of such tasks are: a regular performance check, creation of a new user or a re-start in the case of problems, but also the signaling of a critical condition.

❖ ❖ ❖

If the administrative interface is a GUI, many of the standard administration tasks can not be automated. Repetitive tasks have to be completed again and again, which leads to a high frustration of the administrators. It also can be hard to get remote access to such a GUI.

Unexpected usage. System administrators have their own ways of organizing their administration tasks. The strive to automate many parts, often in unexpected ways, and a GUI is minimizing the possibilities of doing so.

Admin OS vs. System OS. The operating systems which admins are using for their administration tasks often differ from the OS the application to be administered is running on. Providing an GUI as administrative interface often means that this GUI is only executable on certain OS's, which certainly restricts system administrators in an unnecessary way.

❖ ❖ ❖

Therefore: Provide an API for all required administration functionality. Make this API available, easily accessible and well documented, so that admins can automate administrative tasks and integrate it easily in the administration processes.

Offering an API for the administrator provides much more flexibility to the system administrators for administering the systems in the way they think fits best. It gives them enough freedom to integrate the administration in existing processes.

Tools for automation can make use of the administration functionality if they can connect to the provided API. The right API helps to automate tasks that are part of a new employee account creation process. [Limoncelli 2011].

❖ ❖ ❖

Rationale.

Using this pattern will make the life of a system administrator more pleasant, especially in cross-platform situations, as it removes the platform-specific issues caused by a graphical administration interface. In combination with a cross-platform scripting language (e.g. Python, Ruby, TCL) this pattern shows its real strength as one can uniformly approach the administration API on any given platform.

SINGLE FILE LOCATION

The application files, e.g. configuration files, data files, or binaries, are stored in predictable and consistent location.

❖ ❖ ❖

If the application files are spread over different folders or hidden in system-folders of the OS, it is hard to keep track of the different locations and to retrieve the correct files. It is hard to achieve a sufficient quality of versioning when the files that should be under version control are spread over the whole disk.

Distributed Applications. Many applications consist of different subsystems, which often require subsystem-specific administration tasks. These subsystems are in many cases developed by different teams, resulting in separated groups of similar artifacts for each subsystem. This makes the organization of the development process more easy, as the teams can work mostly independently.

Hard-coded Locations. Many developers tend to choose a good place for configuration files and make this path relative to the location of the application installation. If this path is hard-coded it cannot be changed other than by building and deploying a new version of the application.

❖ ❖ ❖

Therefore: Put all related files in one (hierarchical) location. Make the path of this location configurable.

Files that logically belong together and should be at the same location are: the binaries of a system, the configuration files and the data files. In the case of log files one should first consider to USE BUILT-IN SYSTEM LOGGING.

❖ ❖ ❖

Rationale.

Without using this pattern the files of applications will be dispersed over several distinct locations which makes it hard to maintain the application. When a file of a module isn't used anymore it will easily remain in disuse and get overlooked which causes pollution of your hard disk.

USE BUILT-IN SYSTEM LOGGING

The application needs to provide the ability of logging certain events or actions by using the built in system logging of a platform.

❖ ❖ ❖

Having a variety of logging formats and log-file locations makes it hard to monitor the state of a whole enterprise, including all running applications.

Format Variety. A high variety of logging formats increases the complexity of integrating the information held within those several log files. It becomes a burden to nullify the different lay-outs of these log files.

Location Variety. When having a variety of log file locations the dispersion of those locations makes it difficult to gather those files to one stack.

❖ ❖ ❖

Therefore: Use the built-in system logging mechanism or define a standard format to be used by all systems.

Don't reinvent the wheel. Many monitoring tools use the system built-in logging mechanisms so don't try to circumvent it. This allows the administrators to make use of existing tools that collect, centralize, and search the logs [Limoncelli 2011].

If it is not possible to use the built-in system logging, e.g. because of different operating systems being used, then define a standard for your system landscape and ensure that this is used for logging. Combine this approach with SINGLE FILE LOCATION.

❖ ❖ ❖

Besides the above mentioned reasons, it is a lot easier to automatically generate incidents from specific defined events from the built-in system log for an IT service management (ITSM) tool. This ITSM tool can be configured to forward the automatically generated incidents directly, without human intervention, to the second line specialists.

CENTRALIZED IDENTITY MANAGEMENT

also known as: IDENTITY MANAGEMENT BUS.

The system makes use of user identities which need to be managed.

❖ ❖ ❖

Decentralized user identity management means a lot of extra work as identities have to be managed on many different places and it is hard to get a centralized overview of all existing or available identities. This also makes role management much more complex.

Separation of Duties. Especially when Separation of Duties (SoD) is a concern such as within financial environments it is important for organizations to be able to show to e.g. an EDP auditor that all regulations are fulfilled.

❖ ❖ ❖

Therefore: Make use of a centralized identity management system if this is available.

This solution has several advantages: if the centralized identity management system (CIM) is also connected to the human resources system (HRM)-system, it is easier to revoke certain grants due to retirements etc. Also user roles could be (automatically) inferred from function profiles in the HRM system.

When no centralized identity management system is available a lot of organizations make use of something like Active Directory Services (ADS). Mostly in these cases where ADS is used this isn't connected to a HR system whereby the events or triggers for the HR processes placing in, leave service, function change or department change are missed in the ADS. This causes an increase in maintenance activities to take care of pollution of the ADS.

❖ ❖ ❖

There is an urgent need within medium to large organizations to centralize role based access information. Several applications have role based access information. This dispersion of information leads to a high maintenance sensitivity. Which demands a high level of deployment. Therefore the dispersion of information needs to be centralized within a solution according to this pattern.

3. ACKNOWLEDGEMENTS

todo

REFERENCES

ALEXANDER, C., ISHIKAWA, S., AND SILVERSTEIN, M. 1977. *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series)*. Oxford University Press.

BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., AND STAL, M. 1996. *Pattern-oriented Software Architecture - A System of Patterns*. John Wiley & Sons, Chichester.

HANMER, R. 2007. Patterns for Fault Tolerant Software.

HARRISON, N. B. 2011. Phd. Ph.D. thesis, Rijksuniversiteit Groningen.

LIMONCELLI, T. A. 2011. A plea from sysadmins to software vendors: 10 Do's and Don'ts. *Communications of the ACM 54,* 2, 50–51.

POORT, E. R. AND VLIET, H. V. 2011. Architecting as a Risk- and Cost Management Discipline. *2011 Ninth Working IEEE/IFIP Conference on Software Architecture 85,* 9, 2–11.

SCHUMACHER, M., FERNANDEZ, E., HYBERTSON, D., AND BUSCHMANN, F. 2005. *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons.