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

TODO

Categories and Subject Descriptors: K.3.2 [**Computers and Education**]: Computer and Information Science Education —*Computer science education*

General Terms: Collaborative Learning, Education

Additional Key Words and Phrases: Educational Patterns, Design Patterns, Teamwork

ACM Reference Format:

Bijvank, R., Köppe, C., Wiersema, W. 2013. Software Architecture Patterns for System Administration Support – L(October 2013), 0 pages.

1. INTRODUCTION

These patterns are inspired by 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 [**?**]. Refer to POSA [**?**]. Describe gap between software architecture and system (or better application?) administration.

Focus of Software Architecture is often on realizing varying quality attributes, as described in eg. ISO 25010: Functional suitability, Performance efficiency, Compatibility, Usability, Reliability, Security, Maintainability and Portability. Many patterns have been described and their applicability for realizing the qualities has been discussed (**TODO**: cite e.g. Harrison/Avgeriou for general (WW, heb geen mooie publicatie van Paris die heirover gaat kunnen vinden), also Hanmer for fault tolerant systems [**?**], Eduardo Fernandez for Security Patterns, look for more ISO-related ones). But there is one quality attribute where not much attention has been paid to: Portability and its sub-qualities Adaptability, Installability, and Replaceability. **TODO**: make this list more precise (WW: removed conformance => not part of ISO25010)).

There have been several initiatives to describe infrastructure and middleware patterns from the perspective of a system administrator:

—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 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*).

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

—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.

But both initiatives don't touch the intersection of software architecture and system administration. Therefore we want to introduce a set of patterns which bridges this gap. The problems that are cited in the aforementioned article have been experienced within daily system administration practice. With these patterns we want to give a handle to application developers to improve their applications from a system administration viewpoint.

Further Ideas

—Hergebruik Zorg voor een standaardbibliotheek van standaard gebruikte oplossingen. Bijv. Logging, Security, Datumafhandeling, format van veelgebruikte user-data. Beheerders ondervinden veel last van dat voor iedere applicatie dezelfde functionaliteit toch net even anders gerealiseerd is en anders geconfigureerd. *Comment Christian: dit zou in een pattern gaan richting* STANDARDIZED ADMINISTRATION TOOLS *ofzo, dit zou dan wel* USE BUILT-IN SYSTEM LOGGING *kunnen vervangen of specifieker maken.*

—Goede functionele beschrijving. Redelijke beschrijving van wat een systeem globaal doet, uitgebreid met technische details rondom de werking. Waar draait de applicatie, waarom is deze applicatie nodig? Welke interfaces zijn er, wie gebruikt deze interfaces? Configuratie instellingen. *Comment Christian: dit staat ook in het CACM artikel. Maar waarom is dit belangrijk, dus wat is het probleem als je het niet doet?*

—NFR's Hoeveel verkeer, hoe lang mag een transactie duren? Welke patterns worden gebruikt: synchroon/asynchroon. Heldere QoS/SLA: Beschikbaarheid, support, onderhoudsvensters, audit en logging. *Comment Christian: Hier weet ik nog niet in hoeverre dat binnen de scope van dit paper valt. Kunnen we morgen bespreken.*

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 [**?**]. 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 offer the possibility of different administration possibilities, as e.g. a regular performance check, creation of a new user or a re-start in the case of problems, but also the signalling 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 and easily accessible, so that admins can automate administrative tasks and integrate it easily in the administration processes.

Offering an API for the administration 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 e.g. helps to provide a service automatically as part of a new employee account creation system [**?**].

❖ ❖ ❖

Rationale.

SINGLE FILE LOCATION

The application is physically distributed over a couple of different files. These could be configuration files, data files, or multiple binaries.

❖ ❖ ❖

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 find the correct files back. A sufficient versioning is not easy to achieve when the files that should be controlled are spread over the whole disk.

Distributed Applications. Many applications consist of different subsystems, which often do require some 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 making a new version of the application.

❖ ❖ ❖

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

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

❖ ❖ ❖

Rationale.

USE BUILT-IN SYSTEM LOGGING

The application needs to provide the ability of logging certain events or actions.

❖ ❖ ❖

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.

Forces

Having a variety of logging formats makes it difficult to integrate the information held within those several log files and to nullify the different lay-outs of these log files. When having a variety of log file locations the dispersion of those locations makes it difficult to gather those files to one stack (too much rephrasing !!!).

❖ ❖ ❖

Therefore: If possible, use the built-in system logging mechanism.

This allows the administrators to make use of existing tools that collect, centralize, and search the logs [**?**].

❖ ❖ ❖

Rationale.

Don't reinvent the wheel. Many monitoring tools use the system built-in logging mechanisms so don't try to circumvent it.

Beside 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 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.

Forces

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

❖ ❖ ❖

Therefore: If possible make use of a centralized identity management system.

Solution Description

Advantages: if connected to HRM-system, revoking grants can be related to retirements etc. Roles could be (automatically) inferred from function profiles in the HRM system.

Relation with possible CENTRALIZED ROLE MANAGEMENT and ROLE BASED ACCESS CONTROL (RBAC) and at a lower level ACCESS CONTROL LIST (ACL)?

❖ ❖ ❖

Rationale.

There is an urgent need within medium to large organisations to centralise 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 infor ... Many organisations have several identity sources, e.g. HR-system, Active Directory, access management systems (physical).

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.

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.