INFORMATIK 2011 - Informatik schafft Communities
41. Jahrestagung der Gesellschaft für Informatik , 4.-7.10.2011, Berlin
www.informatik2011.de

Software Configuration Management in the Context of BPM and SOA

Jörg Hohwiller
Capgemini, Offenbach
joerg.hohwiller@capgemini.com

Diethelm Schlegel
Capgemini, Offenbach
diethelm.schlegel@capgemini.com

Abstract: *Service oriented architectures* (SOA) have established to shape large IT landscapes. *Business process management* (BPM) aims to bring more flexibility to the enterprise and pushes a business driven SOA. In the last years *BPM-suites* (BPMS) have grown to large and powerful systems. They both address development (modelling) as well as execution. Hence they have a big impact on the *software configuration management* (SCM). This paper consolidates best practices that help to do professional software configuration management in the context of BPM and SOA.

1 Problem

Software configuration management [SBG+05] is a quite old and established domain in the context of software engineering. However SCM is a moving target as new IT trends sometimes have impact on it. As we will see this massively applies for a service-oriented architecture when cloud-services and BPM come into play. Classical SCM has a strong focus on the engineering of a single product. Within a SOA these approaches can be applied to individual domain- or service-applications. However for the IT landscape as a whole a gap remains especially for *deployment* and *release management*. Additionally a rich BPMS is both development and execution environment leading to a mixture of aspects that are typically separated. In BPM projects the process models and business rules become central artefacts of the development with high impact on *version control*.

An example architecture for BPM on enterprise level is illustrated in figure 1. Specific products like a BPMS and an enterprise service bus (ESB) have a central and important role in this architecture.

In this context new problems arise for SCM that can be identified by the following questions:

- What happens to version control if functional changes are performed directly inside an ESB or the BPMS rather than in code?
- How can branches be handled for artefacts like process-models or business-rules?
- How to manage (test-)environments in this context?
- What impact does SOA and BPM have on release- and change-management?

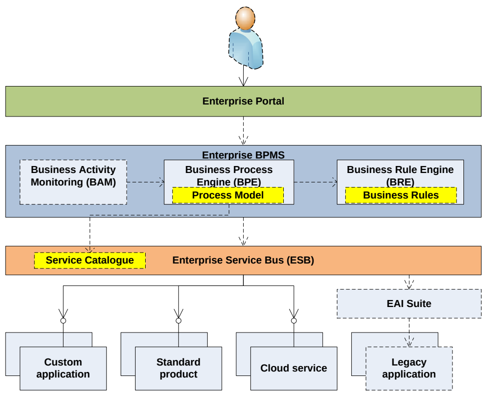

Figure 1: Example Architecture

Generic SCM literature such as [BA08] and [BMT99] do not address such questions but without answers a BPM-driven SOA-project will run into trouble. As an example imagine the impact on quality if process-models are directly modified within the productive BPMS or if services are changed but running process-instances still expect the services to be unchanged.

2 Solution

As a solution this paper provides best practices for SOA-based BPM projects structured by relevant disciplines of SCM. For further details on SCM and its domains and disciplines see [Kre09]. We will focus on the aspects addressed in section 1 that are mainly derived from experience and challenges of projects at Capgemini. The following sections examine this impact and give suggestions on how to cope with the situation.

2.1 Version control

Products supporting BPM, BRM, EAI and SOA not only address the execution but also the creation of functionality. This includes the following artefacts:

- service catalogues
- business process-models
- business rules
- UI forms and flows

While software engineers are familiar with the development of source-code using common IDEs and version control products, a new dimension opens here. Enterprise BPMS products act as IDE for the development of business process-models and rules and also perform version control of these artefacts. Therefore the software configuration managers need to *check the capability of the version control* offered by products such as the BPMS. While few products have sophisticated features such as branching, merging and graphical diffs of process-models others are very limited. At the beginning of the project a *clear strategy* has to be defined how to deal with the version control and which tools should be used how. At least when it comes to *managing multiple branches* it can make sense to manage exports of the artefacts in regular version control systems and align them with the regular deployment.

A key success factor for SOA and BPM is that the business is the major driver and the IT assists this. Therefore different stakeholders including experts of the business departments may work collaboratively on process-models and business-rules. Then some of these people typically have little experience with SCM. Have a look on their technical affinity and plan some *coaching* as required to ensure success. Also developers that are used to have a local development environment where they perform changes in isolation have to learn new strategies. At least some check-out and check-in mechanism needs to be supported by the BPMS to reduce the impact of (experimental) changes. If possible developers should also have a local installation of the software stack including the BPMS on their development machine (see section 2.2).

Also *keep things simple*. Instead of switching branches within the same tool, establish different installations of the products (see figure 2). This may seem awkward as you would never have multiple installations of a regular *version control system* (VCS) like subversion for the same codebase instead of using branches. However, non-IT project members will easily understand that the maintenance of the current production is done in one installation and the development of next major release in another. As a trade-off merge efforts will rise and should not be underestimated, especially when the product has poor support for this. In this context the *number of branches* should be kept at a minimum. A key success factor is to go *smaller steps* and do faster iterations.

We can summarize our best practices on version control as following:

- define a clear strategy for branching and merging
- for collaborative work use different installations for parallel branches
- keep the number of branches low

2.2 Deployment management & environments

As usual at least the environments development, test, acceptance and production (DTAP) should be separated for quality assurance. In regular application development, code is produced and tested in the development environment. After an initial maturity a controlled set of this work is deployed to a central test environment. When the software is feature complete and well tested, it is deployed to an environment where acceptance tests can take place. After the test succeeded the roll-out can start and bring that release into production.

While it is desirable to have many environments and to make them as close as possible to the production environment this is a matter of costs. Especially for SOA and BPM a perfect test- and acceptance-environment would be a copy of the entire enterprise IT landscape. This includes costs for hardware, operation and maintenance as well as licensing costs. From an economic point of view it gets obvious that some compromises have to be made.

In any case it is required to have a realistic environment that allows to test changes before they go live. With this in mind you should never buy a cloud-service that does not include separate test-instances. For commercial products it is often possible to get licenses for test-environments for free or at least as cheap add-on. Using virtualization and private cloud technology it is possible to realize more environments with less hardware as test- and acceptance environments are not permanently used. However this implies some organizational coordination e.g. planning and announcements of performance-tests if additional resources are required.

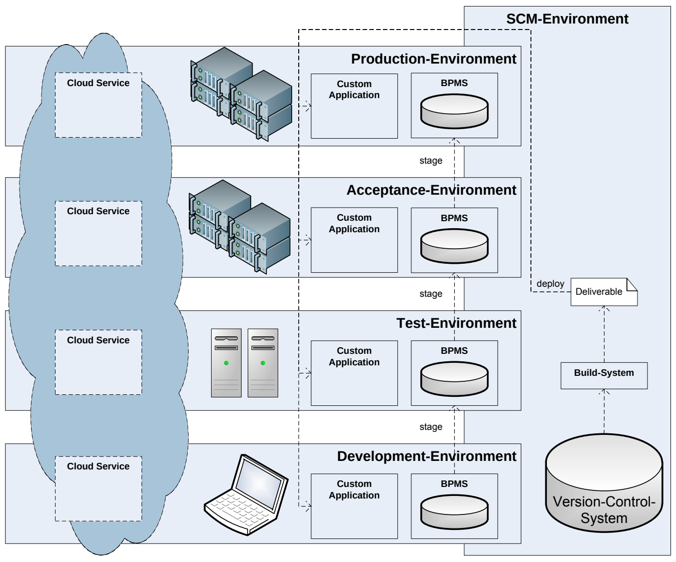

Figure 2: DTAP Environments

The impact on deployment and environments discussed so far is mainly a matter of sizing common approaches on a larger scale. Now if artefacts like process models are directly developed and versioned inside a BPMS new aspects come into play. For collaborative modelling the development environment or at least parts of it may move from the developers machine to a central installation typically accessed trough a web-browser. Additionally if the BPMS is also part of the SCM-Environment (see figure 2) but has installations per environment, changes may or will also happen outside of the development environment.

In classical SCM you will have a central build-management that generates deliverables out of VCS tags and these deliverables get installed in according environments. Here a *staging process* is required for the BPMS allowing to transfer a proper version of the artefacts from one environment to the next (see figure 2). In this case *governance* aspects especially for the deployment to the production environment have to be considered and require new solutions. If the product does not directly support staging, export and import mechanisms need to be used instead. The exported artefact(s) can also be managed via the regular VCS to gain more control over the deployment.

A very important issue is that *environment specific configurations* remain unchanged by this staging process. While there are classical solutions for custom software, in case of standard products like a BPMS you have to deal with the offered possibilities of the product. Ideally parameters like service-URLs of your SOA only differ by a prefix from one environment to another and the BPMS product allows to define and use central variables for such purpose. The goal is to make the staged artefacts including process models environment independent. However the environment specific variables have to be excluded from the staging process. For instance IBM WebSphere Lombardi Edition addresses this by a central table of these variables with values for all environments. Therefore the only environment specific configuration is an administrative property telling if an installation is a production, acceptance, test or development environment.

In any case each environment should be *separated on network level* to prevent accidental access between different environments (e.g. test-environment invoking productive services)

For deployment management our best practices are as following:

- create separate environments and be aware of changes outside development environment
- establish a staging process between the environments
- consider governance aspects for the deployment
- separate environment specific configurations in products like BPMS
- deny network access between environments

2.3 Release management

Release management in context of SOA and BPM is a challenge of its own. It has to be aligned properly with the entire change-management of the enterprise.

In a complex enterprise application landscape release cycles of the individual applications can not be aligned and fixed exactly. Otherwise the delay of a single project would defer the go live of all other changed applications. Hence for each application releases need to be planned in a way that upcoming changes of other applications can be *rolled out as independent as possible*.

This gets even more important in the context of BPM and SOA when it comes to long lasting processes. If a bigger change requires the modification of service-interfaces, it is not enough to change these service-invocations in the process-models. Process-instances that have already been started before the roll-out of the new release continue to run with the previous version of the process-model. Therefore such process expects the services to behave as they used to do in the previous release.

In general a service-interface should be designed in a very stable way. However it is impossible to prevent changes at all. One solution is to *add versions to services* and their end-point-address and always include a service adapting at least the previous version. An alternative is offered by the approach of *normalized systems* that allow to add and remove attributes in such a way that the API does not break [MV09].

We sum up the best practices for release management as following:

- plan applications for independent roll out
- consider running process-instances when processes are updated
- do versioning of services or design them as normalized systems

3 Conclusion

We have seen that SOA, BPM, and cloud computing bring new challenges to SCM. For relevant SCM disciplines the impact has been discussed and best practices have been summarized. Combining existing SCM experience with the solution presented in this paper offers a strategy that addresses the challenges.

References

[BA08] S. Berczuk and B. Appleton. *Software Configuration Management Patterns: Effective Teamwork, Practical Integration.* Addison-Wesley, Boston, Mass., 10. print. edition, 2008.

[BMT99] W. Brown, H. McCormick, and S. Thomas. *AntiPatterns and Patterns in Software Configuration Management.* John Wiley & Sons, 1999.

[Kre09] M. Kreidenweis. Software Conguration Management in Centralized and Distributed Custom Software Development. Master's thesis, University of Augsburg, 2009.

[MV09] H. Mannaert and J. Verelst. *Normalized Systems - Re-creating Information Technology Based on Laws for Software Evolvability.* 2009.

[SBG+05] J. Scott, J. Brannan, G. Giesler, M. Ingle, S. Jois, et al. *828-2005 - IEEE Standard for Software Configuration Management Plans*, 2005.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html