Business Objectives Compliance Architecture Framework

Mechanisms for Controlling Architecture Artefacts

Christoph Moser
BOC Information Techn. Consulting AG
Wipplingerstr. 1
A-1010 Vienna
Austria
christoph.moser@boc-eu.com

Mathias Winklhofer
BOC Information Systems GmbH
Wipplingerstr. 1
A-1010 Vienna
Austria
mathias.winklhofer@boc-eu.com

Christian Kuplich
BOC Information Techn. Consulting GmbH
Voßstr. 22
D-10117 Berlin
Germany
christian.kuplich@boc-de.com

Abstract: At present, more and more IT organisations adopt new architectural practices to effectively deal with the increasingly heterogeneous landscape of their IT architectures. The efficient and integrated management of the various architecture artefacts and their interdependencies - described in architecture models - is a crucial success factor for these initiatives. This paper presents an approach for ensuring architecture artefacts (as the main work products of enterprise architecture management) are under appropriate control. Due to the emerging complexity of IT architectures, the huge amount of crucial information as well as the requirement to ensure conformity to regulations such as SOX and Basel II, the authors view this as a field in need of an approach in order to control and manage this complexity.

1 Introduction

While Enterprise Architecture Management (EAM) frameworks like TOGAF [TO06], Zachman framework [ZS87], DoDAF [Do07] etc. provide guidelines for the designing, planning, and implementation of EAM, they currently do not specify approaches for controlling the various architecture artefacts. In fact they often only provide a rather abstract view on the enterprise, without taking all the required architectural levels into account. An example of a metamodel that consistently specifies the needed architecture artefacts on different layers and in different views is presented in [Bo07]. However, concepts for controlling Architecture Artefacts with regard to their state, validity in time as well as concepts for versioning are not detailed.

Not least because of regulations such as SOX, Basel II and Solvency II, IT organisations are now aiming for increased maturity in their processes and to align themselves to international standards and best practices such as, COBIT [IT07], ITIL [OG07] and CMMI [CM06]. Common ground for each of these standards is the claim for placing their various architecture artefacts under appropriate levels of control. Therefore each of these de-facto standards defines a configuration management process e.g. for managing source code within the development process („CMMI for Development“) or for managing configuration items (such as applications, software components, infrastructure figuration items (such as applications, software components, infrastructure and communication elements) within an IT infrastructure („CMMI for Services“, ITIL, COBIT).

However, these standards do not provide adequate concepts for controlling the various architecture artefacts in a way that is suitable for supporting these processes and practices. Mechanisms, e.g. to organise version control of architecture artefacts, to establish and maintain time-related views as well as mechanisms for status management of the architecture artefacts, remain unattended. As COBIT and CMMI do not discuss these mechanisms at all, ITIL introduces the concept of the configuration management database (CMDB) - renamed to configuration management system (CMS) in ITIL V3 – a repository of all authorised configuration items. ITIL states, that the main purpose of the CMDB is, the provision of up-to-date and secure information via the configuration items used to support all service management disciplines [OG07]. By contrast, the main focus of EAM, besides representing the as-is IT infrastructure, is the creation of planning scenarios, comprising different variants of implementation options, as well as the agreed target IT infrastructure. Since there seems to be a major overlap with regard to the required metamodels, as well as the control mechanisms between EAM and Configuration Management, equal mechanisms and tool functionality seem to be applicable.

This paper introduces a framework called BOCAF (Business Objectives Compliance Architecture Framework), which provides a means for integrating the requirements of both EAM and Service Management. The main focus is on the description of the underlying mechanisms to ensure architecture artefacts are under an appropriate level of control. The term “architecture artefact” is used synonymous with the term „configuration item“ as the elements of the EA are denominated in ITIL. This would guarantee consistency between the two domains. BOCAF is realised within the metamodelling tool ADOit and has proven to be of value in numerous projects. The underlying research approach for developing the BOCAF mechanisms was „Design Science“ [He04]. The discussed mechanisms evolved step by step during the various EA projects conducted by the authors.

The remainder of the paper is organised as follows. In chapter 2, BOCAF and its main elements are explained, by integrating the de-facto standard ITIL and the EA framework TOGAF – as some of the most recognised approaches in the fields of service and EA management. Chapter 3 discusses the various control mechanisms for maintaining and evaluating the EA. In chapter 4, the various mechanisms are brought together, describing the interplay of these mechanisms to support effective and efficient EA and service management. Finally, chapter 5 gives you an outlook on our future research fields.

2 BOCAF – A Framework for Business Objectives Compliance Architectures

BOCAF represents a technology independent EAM framework for the structured and consistent construction and maintenance of blueprints for the management of IT. It allows the comprehensive coordination of strategic positioning, the reorganisation of organisational structures as well as the IS design on various levels by providing three essential elements:

- the metamodel, comprising ITIL's configuration items, as well as further architecture artefacts like reference patterns and the architecture building blocks used during the architecture development process,
- best practice processes for implementing and performing IT management and
- mechanisms for putting the architecture artefacts under control, as well as mechanisms for the analysis and evaluation of the architecture.

Unlike the EA frameworks mentioned above, BOCAF comprises application-oriented scenarios for managing IT. It represents a collection of frameworks and approaches that can be generally accepted as best practice for IT service management and EAM. It allows for integration of different frameworks depending on the particular application scenario. [MB07] for example describes an approach for increasing the maturity level of IT processes in accordance with ISO 20000, a prominent standard in the field of service management by integrating ITIL and CMMI.

The underlying mechanisms for assuring an appropriate level of control for architecture artefacts remain valid for all application scenarios, taking into account that different levels of control are appropriate for different architecture artefacts at different implementation states (e.g. planned, in test, released) at different points in time.

2.1 Metamodel

The language for representing the architecture artefacts and their interdependencies (the model) is described by its metamodel, i.e. the metamodel is a model of its corresponding modelling language [Ku03]. The metamodel of BOCAF consists of a layered architecture, and contains more than fifty modelling classes, with 14 static diagram types (representing the layers). For a more detailed discussion of BOCAF's standard metamodel refer to [Bo07].

The *Strategy* layer supports the alignment of the business targets with those of the IT organisation. Strategies are implemented in terms of *IT Projects*, representing strategic measures. By determining the required projects and by synchronisation of the superordinate project master plans, these measures are assigned to the various architecture levels of BOCAF [MB07]. Figure 1 depicts the various layers of BOCAF.

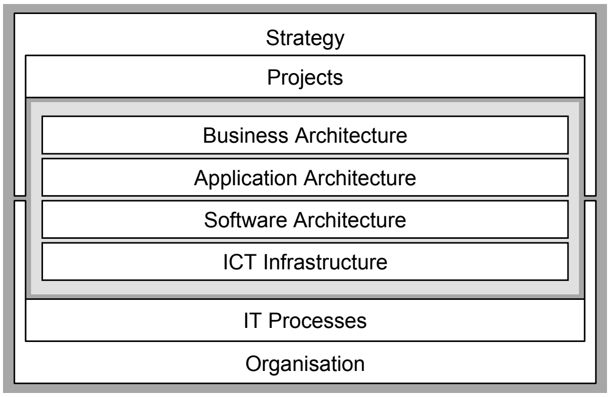

Figure 1. **The BOCAF Metamodel (Overview)**

The *Business Architecture* layer is used to analyse the business processes that are necessary for IT support. *IT services* (also part of the *Business Architecture*), as the products of the IT department/organisation, represent the link between business processes and the existing IT architecture. The *Application Architecture* layer represents the applications, their interfaces and business functions. Each application can be detailed in the *Software Architecture* layer, comprising the deployment units of the applications, interfaces as well as information and data flows. (Of course interfaces, information and data flows are part of the *Application Architecture* layer as well.) For running the necessary applications, infrastructure elements are structured in the *ICT Infrastructure* layer, representing all infrastructure and communication elements of the various test and production environments. The *IT Processes* and *Organisation* layers represent the process organisation and the organisational structure (including roles and skill profiles) of the IT organisation.

2.2 Best Practice Processes

Some of the aforementioned EA frameworks provide reference models, which represent approaches to developing and maintaining the EA. These reference models can be integrated into BOCAF on its *IT Processes* layer. Table 1 gives an overview of an integrated process model combining ITIL and TOGAF's Architecture Development Method (ADM), which builds the basis for deriving the necessary mechanisms discussed in chapter 4. It is worth mentioning that the described process model, focuses on the processes that are in need of mechanisms for controlling the architecture artefacts and maintaining the EA in an efficient way. It does not cover all of the processes recommended in ITIL and TOGAF. For a more detailed mapping considering all relevant processes, refer to [Th07].

Process	**Short Description**	**Required Mechanisms**
Analyse the EA	Whereas the analysis in service management concepts often focuses on the improvement of single IT services (e.g. Availability Management); in EA the analysis usually focuses on the capabilities of the IT architecture to meet the business goals in the long run.	Groundwork for the analysis builds a repository, representing the architecture at any given point in time, building the baseline for further planning (see chapter 3.1). The concept of „Types and Instances“ (chapter 3.2) ensures the efficient documentation of the architecture with the required level of detail.
Propose Changes	For proposing changes we find elements in both fields. TOGAF distinguishes three change types: The “simplification change”, much like the definition of change in ITIL. The “Re-architecting change”, which requires putting the whole architecture through the architecture development cycle again. The “Incremental change” may be capable of being handled via change management techniques as discussed in ITIL, or it may require partial re-architecting.	An Request for Change (RFC) represents an architecture artefact, containing a description, objectives, description of the risks and effects of not implementing, and further attributes (all part of the metamodel). RFCs need to be approved via an agreed release workflow (see chapter 3.3).
Generate and Analyse Planning Scenarios	For “Re-architecting changes” and for “Incremental changes”, because of their complexity, the definition of scenarios might be appropriate. These scenarios need to be analysed with regard to their capabilities to meet the business needs. As discussed in [MB05] an agreed technology roadmap, e.g. derived from a Technical Reference Model (TRM), as provided by TOGAF or FEAF should build the construction kit for standard conform scenarios.	A scenario mechanism to describe the implementation options and their interplay with the baseline architecture (at a certain point in time) is requested (see chapter 3.1). The competing planning scenarios need to be connected to the initiating RFC via relationships as defined in the metamodel. Once the most applicable scenario is chosen, its architecture artefacts need to be released into the baseline section of the repository (see chapter 3.3).

Perform Project Master Planning and Migration Planning	This process is about generating an overall implementation and migration strategy and a detailed implementation plan for the implementation of the released artefacts. For this purpose ITIL establishes the concept of the Forward Schedule of Change, which contains details of all approved changes and their planning. TOGAF uses an equal concept relating to a time-lined implementation roadmap.	Impact analysis capabilities to identify changes/projects working on the same instances of architecture artefacts at the same time are needed. Furthermore, interdependencies in terms of budget and resource restrictions need to be avoided (see chapter 3.1).
Perform Implementation Governance	The goal of this process is to ensure that changes to the architecture are managed in a cohesive and architected way [TO06]. Parallel with this phase there is the execution of an organisational specific development process, where the actual development happens [TO06]. ITIL introduces the process "Release Management" for this purpose.	Check mechanisms that ensure the validity of the released artefacts must be in place. This implies the necessity of mechanisms to comprehend the relations of CRs, planning scenarios, released artefacts, scheduled changes/projects, as well as to verify the content of the repository against the real world environment (see chapter 4). The approach for verifying artefacts is taken from the process "Verification" of ITIL's Configuration Management process.

Table 1. **Considered processes of ITIL and TOGAF**

2.3 Mechanisms

As stated above, two major types of mechanisms are implemented in BOCAF:

1. Mechanisms for the analysis and evaluation of the released architecture and planning scenarios comprise:
    - mechanisms for the generation of dynamic views like portfolio view, layer-spanning dependency views, GANTT views, to perform analysis of the EA from numerous concerns (requirements) or viewpoints and to provide means to document each of the relevant viewpoint [TO06], [Bo07] and
    - mechanisms for evaluating and designing the EA, using concepts like CMU/SEI's Architecture Trade-off Analysis (ATA) Method, the concept

of gap matrix etc. [TOG06].

2. Basic mechanisms for putting architecture artefacts of the EA under appropriate control in a way to guarantee validity and timeliness of its content.

The following chapter discusses the basic mechanisms, derived under consideration of TOGAF as well as from the required configuration management concepts of ITIL V3.

3 Requirements of Mechanisms for Controlling Architecture Artefacts

3.1 Baselines and Planning Scenarios

As intended in the step "Opportunities and Solutions" of TOGAF's reference process, model mechanisms for the definition of various planning scenarios need to be provided. Planning scenarios represent implementation options, described by using the various diagram types representing the *business, application, software* and *ICT infrastructure* layers of BOCAF. Examples for planning scenarios are build-versus-buy-versus-reuse options, and sub-options within those major options [TO06]. The various planning scenarios need to be evaluated with regard to their suitability for implementation. After performing a benefits analysis, the most appropriate scenario can be chosen, based on parameters such as costs and amortisation time.

The starting point for a new planning scenario is the architecture at any given point in time, along with a baseline which represents the released, authorised state of the enterprise at this time. The released artefacts comprise artefacts already implemented at this point in time, as well as those that are planned or in a test state (see chapter 3.4). Baselines at predetermined states of the enterprise architecture are formally reviewed and agreed on, and serve as the basis for further development of the designated changes to the architecture.

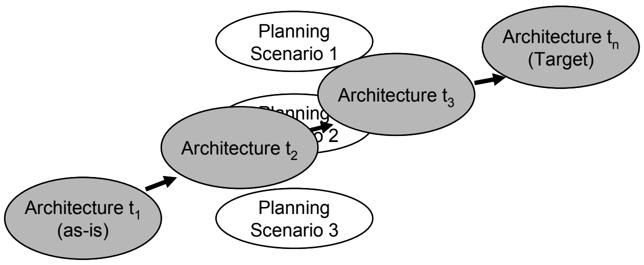

Figure 2. **Baselines and Planning Scenarios**

BOCAF consists of a repository that holds all the architecture artefacts that are part of the enterprise's current IT environment, as well as artefacts that have yet to be integrated. It contains information to support the operative ITSM processes as well as typical information, like reference patterns and architecture building blocks to support the architecture development processes. An important concept is the separation of the repository contents in released artefacts that represent the state of the IT infrastructure as-is and artefacts that are not integrated yet. The latter ones might be created with the purpose of planning to change or extend the current IT infrastructure, but have not been released yet or the planned components are just not installed yet. An advantage of the planning scenario concept is the possibility to create several concurrent planning scenarios, execute queries on the planned architecture and compare the results of *n* possible scenarios.

In BOCAF the separation between actual, released artefacts and artefacts within a planning scenario is made possible by filtering the repository contents in a way that only artefacts within a certain context are visible to the user. Roughly, the repository contents can be separated into artefacts that were released (thus agreed upon and possibly already part of the actual infrastructure) and those that are contained within a new planning scenario. The first are visible all the time but can only be manipulated through the release workflow (see chapter 3.3). Manipulation of the released elements as-is through modelling means, is not possible since they are already in use and possibly needed by other scenarios or running projects. The latter will only be visible when the planning scenario they are part of is activated. The user can still manipulate elements within a planning scenario as long as they have not been released.

Upon release, an architecture artefact does not lose its affiliation to a planning scenario. The process of releasing artefacts is further described in chapter 3.3. Additionally to its release status, an artefact has an implementation status. Releasing an artefact only makes it available for projects, it does not necessarily mean that the released artefacts are already implemented (see chapter 3.4).

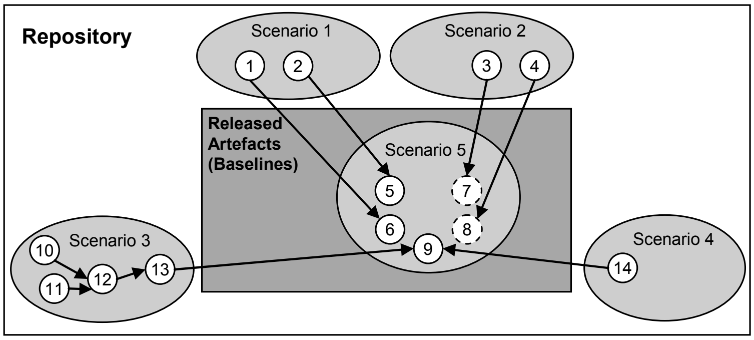

Figure 3. **Planning Scenarios**

Consider the figure 3:

- Scenario 1 consists of the artefacts 1 and 2 that were newly created, but it also uses artefacts 5 and 6 that are already released and are still affiliated to scenario 5, that only contains released artefacts. Scenario 5 was implemented completely.

Example: A new application along with a new database is to be introduced in an enterprise. A month ago a project was finished successfully and introduced three new servers that still have capacity available. For the new application and the database, the existing capacity should be used.

- Scenario 2 consists of the artefacts 3 and 4 and also uses released artefacts. In this scenario, the released artefacts 7 and 8 are not realised yet (which is visualised in the figure by the artefacts' dotted line).

  Example: Similar to scenario 1, new software is to be introduced and installed on existing hardware. This does not pose any problems since there is still capacity available and there are no other scenarios that need any of the resources of this scenario. However, it has to be considered that, although the artefacts 7 and 8 are already released, they are not realised yet (because they are still being tested), and are not available immediately.

- Scenario 3 consists of the artefacts 10, 11, 12 and 13. The architecture artefacts introduced with this scenario will be using the artefact 9 that is already part of the released IT architecture. Another scenario, scenario 4 also plans to use the artefact 9. In this situation, the persons responsible for planning have to be aware that seemingly spare resources might already be occupied in another planning scenario. For these situations BOCAF proposes queries (like impact analysis) that provide a mechanism to identify all possible scenarios in which an architecture artefact is used at a certain point in time, to avoid resource conflicts.
  Example: A piece of hardware is available and has enough resources to host another planned application. However, two scenarios are planning to use the hardware and it is not certain that it can support both.

3.2 Types and Instances

In ITIL, architecture artefacts are classified in types and instances where a type refers to a certain modelling class and an instance to a specific version of this class. The classification of architecture artefacts helps to identify and maintain the usage and status of the items. Typical types according to ITIL are: software products, business systems, system software, servers, mainframes, workstations, laptops, routers and hubs.

An example for the relationship between types and instances in BOCAF is a piece of software that comes in various release versions. Every version of this software is an instance which is identified by an instance-specific versioning attribute, e.g. its version number. The versioning attribute for a piece of hardware could be its serial number. (Of course the versioning can be defined by more than one version attribute of an artefact, but to reduce complexity for the modeller BOCAF focuses on only one versioning attribute in its reference implementation.) An instance always belongs to a type; it can never exist on its own. Figure 4 shows an example of the interplay of types and instances: "Quickref" as the type of an architecture artefact (of modelling class "application”) and its related instances “Quickref 1.0” and “Quickref 2.0”, representing the releases of “Quickref”.

On the other hand, a type can have zero or more instances related to itself. Instances are connected to types via the „is-kind-of“ relation (see chapter 3.3), which does not necessary define an inheritance relation from the type to the instance. In fact BOCAF allows the definition of propagation rules from types to instances or vice-versa. For example the propagation of the attribute “date of validity” from the instance to the type via the function *max (date of validity)* can be denoted, thus setting the value of the type’s attribute to the highest value of the attribute, in all of the instances based on this type.

The separation of types and instances provides an opportunity to design IT architectures on two levels: the type and the instance level.

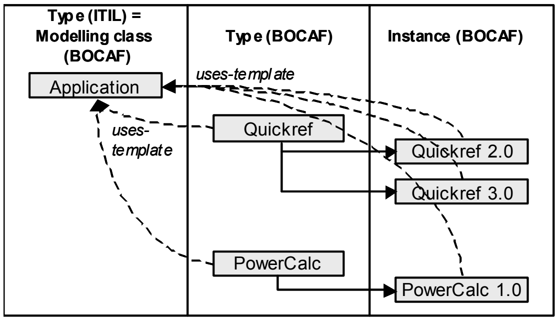

Figure 4. **Interrelations between Modelling classes, Types and Instances**

Figure 4 additionally shows the level “Modelling class” that contains the templates for types and instances as they are defined in the metamodel (mentioned in chapter 2.1). On the type level, hierarchies of any depth are possible. An example would be an abstract application type “Office Software” upon which the types “Quickref” and “PowerCalc” are based. To keep the modelling effort and the complexity of the modelled EA low, the BOCAF metamodel does not contain type hierarchies.

The type level is convenient for modelling dependencies between architecture artefacts, without having to pay too much attention to detail. The instance level is needed when the dependencies are known and the architecture has to be planned in great detail. Also, when modelling on the instance level, it is possible to use the type as a container for several instances to evaluate the architecture. The type represents the version set of all its instances. Thus, it is possible to perform queries on the type level (e.g. “Retrieve all instances of Quickref”) that return all instances of the selected type.

3.3 Versioning Concepts and Release Workflow

Within BOCAF, artefacts and their relations are regarded as versioned items. As discussed in [Sc94] and [CW98] versioning is performed with several intentions:

- A version defined to supersede its predecessor is called a revision. For example, the new version might be the result of a bug fix or the refinement of a previous version.

- Versions defined to coexist are called variants. These represent alternative versions defined by a set of variant attributes. E.g., the metamodel might declare the attribute “target operating system” of the modelling class “application” to be a variant attribute. According to this rule, a variant can be defined for each design alternative with regard to the suitable target operating systems.

- Finally, versions may also be maintained to support cooperation or for analysing different resolution alternatives. This case is covered by the concept of planning scenarios (see chapter 3.1).

BOCAF distinguishes two structural relationships relevant for versioning (in compliance with [Ka90]):

- *Version Histories/Is-Derived-From/Is-A-Kind-Of:* This relation type is used to incorporate the time dimension into BOCAF, by introducing the concept of revisions (see above). Via revisions the ancestor/descendent interrelationships of one artefact are defined (using “is-derived-from” relationships).

  The “is-a-kind-of” relation was elaborated in more detail in chapter 3.2, by introducing the concept of “types and instances”.

- *Equivalencies:* Often a variety of representations are needed to describe architecture artefacts and their dependencies accurately. Therefore mechanisms to tie these representations together are required. Within BOCAF this is resolved via diagram types (for visualising the interplay of architecture artefacts) and the concept of planning scenarios, as previously discussed.

Before a new item that was added to the repository or an existing item that was changed is released and therefore is ready to be added to the actual infrastructure of the enterprise, the relevant item has to go through a release workflow, during which it is reviewed by various persons and finally released.

In BOCAF, the release of architecture artefacts is achieved by bundling them into a planning scenario and sending the whole scenario through a release workflow. The release workflow in BOCAF is realised as a status automaton that defines the states „Draft“, „Audit“, „Released“ and „Archived“.

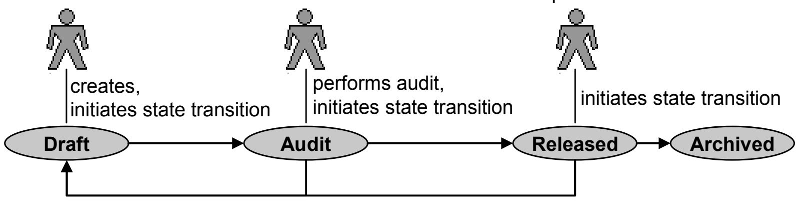
Modeller
Auditor
Supervisor
creates, initiates state transition
performs audit, initiates state transition
initiates state transition
Draft
Audit
Released
Archived

Figure 5. **States of the Release Workflow**

In the figure above, the release workflow consists of the four aforementioned states and three roles that have permissions to initiate state transitions:

- The Modeller creates new diagrams and artefacts in the state „Draft“. These diagrams and artefacts make up a new planning scenario. After the modelling tasks in the new scenario are finished, the Modeller forwards the whole scenario to the Auditor.
- The Auditor decides whether the planning scenario forwarded by the Modeller meets the requirements of the enterprise and can either reject it and send it back to the Modeller for further refinement or accept it and set its status to „Released“.
- The Supervisor manages the process and decides about the modification of the IT infrastructure. To change an existing, released architecture artefact, the Supervisor can create a new version of this artefact and send it to the Modeller who implements the required change. After a new version of an artefact is finished, the Supervisor can set the status of its predecessor to „Archived“. Artefacts that have reached this state are added to the released artefacts within the repository. It is important to note that these artefacts are not necessarily implemented yet, they may still only be planned or in test (see chapter 3.4).

The various versions of an architecture artefact are identified by a numeric version number (extensional versioning). It is important not to confuse the version of an artefact with the concept of instances that was described in the previous chapter 3.2. While for example, the version number of an instance of a piece of software refers to the release version (e.g. for Quickref 3.0, the version number 3.0 would be the number of the release), the version of an artefact in the repository is a mechanism to visualise and manage its level of maturity.

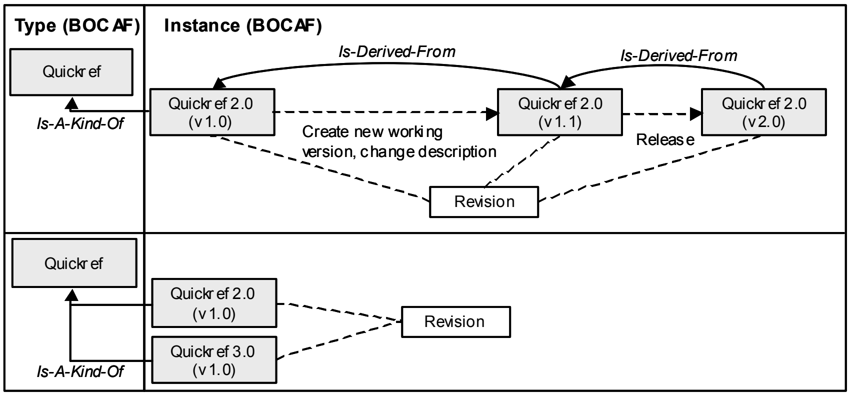

Figure 6. **“Types and Instances” and the Release Workflow**

Consider figure 6 that visualises the difference between the instance of a type (represented by an instance attribute) and the technical version of an architecture artefact. The top of the figure shows how a new technical version of an architecture artefact is created: The artefact “Quickref 2.0” which is an instance of the type “Quickref” exists in its technical version 1.0. After it was released, an error in the description of this artefact was discovered, so it was necessary to create a new working version (Quickref 2.0 in technical version 1.1) and to correct the error. After the error was corrected and the artefact passed the status “Audit”, Quickref 2.0 was released and its technical version was set to 2.0. During the whole process, the artefact’s instance attribute was not changed because the release version of the artefact did not change.

The bottom of the figure shows that a new release of the type “Quickref” is handled differently. It is handled as a new instance that has no actual connection to its predecessor instance (aside from the shared type). The instance “Quickref 2.0” exists in technical version 1.0 side by side with its successor “Quickref 3.0” that also has the technical version 1.0.

3.4 Time-related Views

An important aspect of planning the IT architecture of an enterprise is the capability for describing different states of architecture artefacts at different times. While a piece of hardware might be valid at one point in time, some time later it might have become invalid. BOCAF’s metamodel realises this aspect and incorporates it. In BOCAF, every architecture artefact contains an attribute that represents the artefact’s state for a certain period of time.

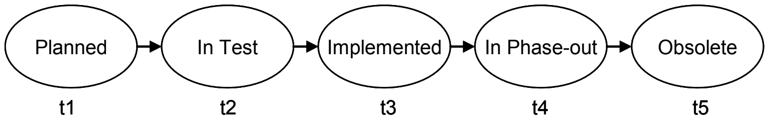

Figure 7. **States of Architecture Artefacts**

The figure above shows the four states that an architecture artefact in BOCAF can have: „Planned“, „In Test“, „Implemented“, „In Phase-out“ and „Obsolete“. Every state also has a date that marks the starting point for the next state. In the figure above, the state „In Test“ would start at t1 and last until t2.

The point of time at which the modelled IT architecture is viewed, affects the visibility of architecture artefacts. Consider figure 8: The first part shows the state of the IT architecture at t1. There are two applications that are using the old server Server1. A new server named Server2 is already planned and will be introduced shortly. The user can now view the state of t2 and see that at this time one application, PowerCalc, has already been migrated to the new server. The other application, Quickref is still using the old server. If the user views the state of t3, he sees that at this time, Quickref has been migrated to the new server as well, the old server Server1 has become obsolete and will from now on be hidden in this diagram. It is important to note that BOCAF provides the possibility to view the state of the architecture (including the artefacts and their relations and dependencies) at any given point in time. According to [Sc94] there are two types of time versions to be considered: Time values in BOCAF are considered as logical time – the time at which the changes took place or will take place in the real world. The time at which the changes took place in the repository - physical time – is not discussed.

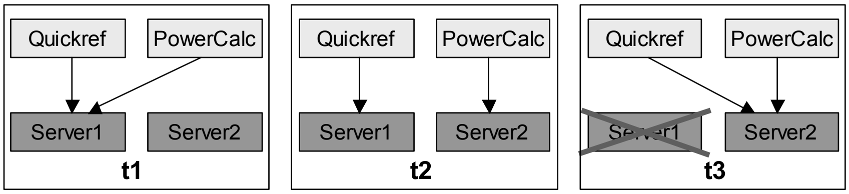

Figure 8. **Example for time-related Views**

4 Tying it all together

Figure 9 illustrates how the various concepts introduced in the previous chapters can be brought together and used to effectively manage the IT architecture. The released architecture artefacts are at the centre of the architecture. They are supposed to resemble the reality as much as possible. A synchronisation of the released artefacts to the reality is performed constantly to minimise differences as proposed in ITIL's Service Asset and Configuration Management [OG07].

While constant synchronisation of the repository and the reality is necessary and sensible, changes made on the repository contents, such as the introduction of a new mail client, will of course affect the real world (by releasing and implementing the mail client in a real world environment), as can be seen at the bottom of the figure.

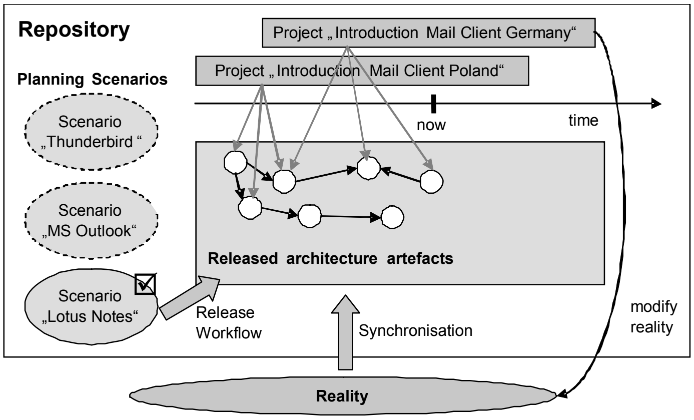

Figure 9. **Interdependencies of BOCAF's control mechanisms**

Planning scenarios are used to introduce new architecture artefacts into an infrastructure. As soon as architecture artefacts are approved and released (as new artefacts or new versions of existing artefacts) in the repository, changes to the infrastructure can be made that are using released artefacts that were introduced through scenarios.

Consider the example in the figure above: On the left side, alternative planning scenarios are listed. In this example, a company wants to introduce a new mail client for its subsidiaries. The company considers three possible clients: Thunderbird, Lotus Notes and MS Outlook. For each of these clients, a planning scenario is created, complete with needed hardware and support services. After all the advantages and disadvantages of the three scenarios have been calculated, a decision for one scenario is made and the required architecture artefacts are approved and released. The process of releasing architecture artefacts was described in chapter 3.3. In the example, the company decides to use Lotus Notes. Only the artefacts needed for this planning scenario are released, artefacts required for the other two scenarios are not further considered and will not be added to the infrastructure.

After the selected scenario has gone through the release workflow, the introduced architecture artefacts (or their new versions) are residing in the repository and are from that time on available within the company.

Above the repository, along the time axis, changes to the released IT architecture are made. This concept represents a time-lined implementation roadmap. In this example, after the architecture artefacts necessary for the introduction of the new mail clients have been released, changes to the company's subsidiaries can be made. First, the new client is introduced in Poland, later in Germany. The status of the released architecture artefacts changes from planned to tested and implemented during the lifetime of the projects/changes. Through projects/changes architecture artefacts might be removed, in this case the state of the architecture object is set to obsolete and its date of validity expires.

5 Experiences and Future Work

The paper provided a general prescription of mechanisms for controlling EAs, which have to be adapted (by IT architects) for the specific problem at hand. For example the concept of "types and instances" might be disabled if it is not required for dealing with a certain situation. This might be appropriate if the architecture work focuses more on a strategic level (by modelling types only) to reduce the planning effort. Furthermore, by building upon a metamodelling platform, the discussed mechanisms work with any metamodel, which can be designed exactly according to the situation at hand.

The usage of the planning scenario mechanisms provides a workspace for IT architects to define and evaluate alternative architecture solutions in parallel, by combining released artefacts with planned artefacts. In doing so, the effort involved in defining independent, consistent alternative scenarios is saved. The concept of "types and instances" is regarded as a powerful mechanism, as it allows dealing with the architecture on different levels of details. This is useful if the architectural work only focuses on certain sections of the EA or detailed planning is not appropriate. Furthermore, the release workflow ensures that accepted scenarios are propagated to the released architecture section, without the risk of losing details of the architectural work. Since the repository provides a view on the EA at any given point in time, it is possible to define projects within the planning scenarios which are based on states of the architecture at any given point in time, not only on certain states which is often a major drawback of conventional modelling tools.

The full power of these basic concepts comes when combined with mechanisms to evaluate the EA – like mechanisms for the generation of dynamic views, graphical and tabular scenario comparisons etc. (Mentioned in chapter 2.3.) Of course the suitability of the presented method needs to be proved in a more structured manner. Our approach will define and evaluate key performance indicators (KPIs), e.g. the "reduction of incidents occurring after changes" reduced through a more credible change process, or indicators evaluating the workload reduction in the various phases of complex EA projects.

Among our future research goals is the definition of a detailed reference model for IT architecture and service management, as well as the extension of the existing technical workflows within ADOit, based on this reference model.

References

[Bo07] BOC: ADOit NP White Paper. (2007).

[CM06] CMMI Product Team: CMMI® for Development. Version 1.2 (CMMI-SE/SW/IPPD/SS, V1.1): Staged Representation, Carnegie Mellon Software Engineering Institute, http://www.sei.cmu.edu/cmmi/models/index.html (access: 2007-08-15).

[CW98] Conradi, R.; Westfechtel, B.: Version Models for Software Configuration Management. In: ACM Computing Surveys, June 1998, Volume 30, No. 2.

[Do07] Department of Defense (DoD): DoD Architecture Framework – Volume 1: Definitions and Guidelines. http://www.defenselink.mil/cio-nii/docs/DoDAF_Volume_I.pdf (access 2007-08-12).

[He04] Hevner, A.R.; March, S.T.; Park, J.: Design Science in Information Systems Research. In: MIS Quaterly, March 2004, Volume 28, No. 1, pp. 75-105.

[IT07] IT Governance Institute: COBIT. Version 4.0, IT Governance Institute, http://www.isaca.org/downloads (access: 2007-06-11).

[MB05] Moser, C.; Bayer, F.: IT Architecture Management – A Framework for IT Services. In: (Desel, J.; Frank, U. Eds.): Proceedings of the Workshop on Enterprise Modelling and Information Systems Architectures, October 2005, Lecture Notes in Informatics, Volume P-75, pp. 137-151.

[MB07] Moser, C.; Bayer, F.: Einführung von ISO/IEC 20000 – Ein prozessbasierter Ansatz. In: (Andenmatten, M. Eds.): ISO 20000 – das Gütesiegel für geschäftskonforme IT-Dienstleister. Publication date: February 2008, Symposion Verlag.

[Ka90] Katz, R.H.: Toward a Unified Framework for Version Modeling in Engineering Databases. In: ACM Computing Surveys, December 1990, Volume 22, No. 4.

[Ku03] Kühn, H.; Bayer, F.; Junginger, S.; Karagiannis, D.: Enterprise Model Integration. In: (Bauknecht, K.; Tjoa, A M.; Quirchmayer, G. Eds.): Proceedings of the Fourth International Conference EC-Web 2003 – Dexa 2003, Prague, Czech Republic, September 2003, LNCS 2738, Springer-Verlag, Berlin, Heidelberg, pp. 379-392.

[OG07] Office of Government Commerce: ITIL – Service Transition. Appeared in the book series ITIL - IT Infrastructure Library, The Stationery Office, London, 2007.

[Sc94] Sciore, E.: Versioning and Configuration Management in an Object-Oriented Data Model. In: (Scheuermann, P.): VLDB Journal, 1994, No. 3, pp. 77-106.

[Th07] Thorn, S.: TOGAF™ and ITIL®. http://www.opengroup.org/architecture/togaf8-doc/arch/ (access: 2007-09-05).

[TO06] TOGAF: The Open Group Architecture Framework "Enterprise Edition". Version 8.1.1, http://www.opengroup.org/architecture/togaf8-doc/arch/ (access: 2007-08-15).

[ZS87] Zachman, J. A.; Sowa, J.F.: Extending and formalizing the framework for information systems architecture. In: IBM Systems Journal, 31. Jg., Nr. 3, 1992, S. 590-616.