Enterprise Architecture Management Patterns – Exemplifying the Approach

Sabine Buckl, Alexander M. Ernst, Josef Lankes, Florian Matthes, Christian M. Schweda
Chair for Informatics 19
Technische Universität München
E-mail: {buckls, ernst, lankes, matthes, schweda}@in.tum.de

***Abstract*—Enterprise Architecture (EA) management has been gaining importance in organizations, and while EA management frameworks provide a holistic and generic view on the subject, organizations introducing EA management are often left alone regarding the details of the approach. The EAM Pattern Catalog, presented in this article, is a collection of best practices for addressing specific concerns in EA management related to e. g. architectural standardization, application landscape planning, or interface, business object, and service management. It provides methodologies for addressing these concerns, together with information models defining the relevant concepts, and viewpoints for visualizing them.**

This article describes the structure and extent of the EAM Pattern Catalog, and exemplifies its approach by outlining EAM patterns for addressing architectural standardization. Architectural standardization tries to tackle the complexity of the EA created by historically grown structures. These structures lead to disadvantages as low maintainability, low bargaining power at IT suppliers, or the need of largely diverse skills in the IT workforce.

I. Introduction & Motivation

Enterprise Architecture (EA) management is one of the major challenges of modern enterprises. It aims at aligning business and IT in order to optimize their interaction. Additionally, regulations like e. g. *Sarbanes Oxley Act* (SOX [29]) require companies to document and plan their EA.

Whatever preliminary work exists in a company, there commonly is a demand for a more structured way to manage the evolution of the EA. A variety of approaches to introduce EA management has been proposed by academia and practice (see e.g. [4], [13], [24]), but they all have to cope with at least one of the following problems:

- EA management is introduced from scratch, not considering related initiatives already present inside or outside the organization.
- EA management frameworks, like Zachman [36], TOGAF [34], etc., are usually either too abstract and therefore not *implementable*, or too extensive to be used in practice, as they have to be utilized as a whole.
- Lacking an actual starting point for an EA management initiative, companies tend to collect requirements from potential EA stakeholders in the organization. Consolidating their demands and integrating their information needs, an all-embracing EA management approach is likely to emerge, which would demand a vast amount of data to be gathered, although only a part of it would be needed to address the real pain points of the company.
- If an approach has been implemented, it is often not documented, why certain decisions have been taken, e. g. why a certain concept has been introduced in the information model. This leads to information models, which cannot be adapted or extended, due to the fact that no one knows what analyses rely on which parts of the information model.
- Approaches proposed e. g. by organizations or standardization groups are usually *all or nothing* approaches, meaning that they are supposed to be introduced as one single piece instead of incrementally. This results in an EA management approach that is not tailored to the company's EA maturity.

In order to address the problems stated above, we propose to apply *patterns*, well known from other disciplines like architecture or software engineering.

Alexander et al. [1] define a pattern in the context of architecture as follows: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." Additionally, [1] details that "each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution".

In software engineering, design patterns are e. g. defined by Gamma et al. [14] as "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context."

The common ground of both definitions is that patterns

- are a general, reusable solution to a common problem and
- are dependent on their context.

The above described properties are the basis for the EAM pattern approach, which initially has been introduced in [8]. EAM patterns describe solutions, based on best practices for recurring problems in EA management that can and may have to be adapted to a specific enterprise context.

In addition to addressing the problems stated above, patterns offer further advantages. According to [16], an advantage of the pattern concept is that it enables architects to understand the impact of the architectural decisions at design time,

because patterns contain information about consequences and context of the pattern usage. Harrison et al. [15] state that patterns have shown to be a useful and important vehicle for capturing some of the most significant architectural decisions. As information on possible consequences of pattern usage is included in pattern descriptions, they qualify for documenting the rationale and expected consequences of a decision. Another advantage is that patterns can also be included in current EA management frameworks to enrich them with concrete instructions on how to address a specific concern.

The following three kinds of EAM patterns have been introduced in [8].

A *Methodology Pattern* (M-Pattern) defines steps to be taken in order to address given concerns. Furthermore, as a guidance for applying the method, statements about the intended usage context are provided, which include the concerns to which the M-Pattern can be applied. The procedures defined by the M-Pattern can be very different, ranging from e. g. visualizations and group discussions to more formal techniques as e. g. metrics calculations [23]. M-Patterns have been introduced, because missing methodologies constitute a common issue in EA management information models. Additionally, frameworks as e. g. TOGAF [34] provide a process model (TOGAF ADM), but leave the details of the methodologies supporting the specific activities in the EA management process relatively open. M-Patterns explicate the methodologies in order to complement activities carried out in an ad-hoc manner or relying on implicit knowledge with activities carried out more systematically.

A *Viewpoint Pattern* (V-Pattern) provides a language used by one or more M-Patterns and thus proposes a way to present data stored according to one or more *I-Patterns*. In our research project *Software Cartography* (see e. g. [7], [8], [22], [35]), we found that industrial users often specify viewpoints by example. This means that an exemplary view is provided for the viewpoint, possibly together with some textual explanations. While we do not contend that this may be sufficient in certain use cases, e. g. sketching concepts in presentations, we see problems arising, when the goal is providing *official* information to a wider audience for an extended period. In order to ensure the understandability of a diagram, we regard e. g. a legend to be mandatory.

An *Information Model Pattern* (I-Pattern) supplies an underlying model for the data visualized in one or more V-Patterns. An I-Pattern contains an information model fragment including the definitions and descriptions of the used information objects. As described in [8], different languages are possible for describing an I-Pattern, varying in their degree of formality, including among others textual descriptions in natural language, the Meta Object Facility (MOF [26]), Unified Modeling Language (UML [25]) class diagrams, ontology languages, and mathematical formalizations, or combinations of these approaches. Choosing a specific approach basically has to consider the needs of the use cases to be supported. While an object-oriented description might be sufficient for creating a visualization or a tabular report, e.g. process simulation may only be reasonably possible on a more formal basis. Therefore, we propose using a language adequate to the problem to be addressed, thereby strongly considering UML as the default language, as it is widely understood and has been found by us to be problem-adequate in many practical settings in the context of EA management information models [7]. In case a language different from UML is chosen, complementing its specification with an UML-based description can yield advantages, especially as integrating information model patterns is simplified by them being available in a common language.

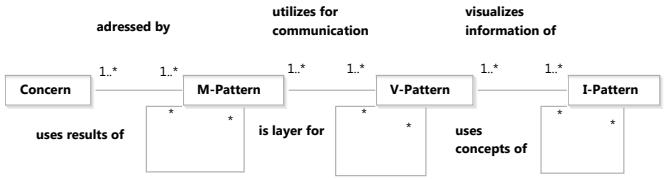

Fig. 1. UML class diagram describing the relationship between Concerns, M-Patterns, V-Patterns, and I-Patterns

The conceptual UML class diagram in Figure 1 shows the dependencies between the EAM pattern types. The class *Concern* in the conceptual diagram reflects the fact, that the EAM pattern approach is concern driven. Concerns are usually the entry point for management activities.

In order to improve readability and comparability, the structure of the EAM patterns is similar to the structure proposed by [14]: pattern name, problem, solution, and consequences. Additionally, all EAM patterns include versioning information in order to be able to differentiate between different development stages.

The remainder of the article is structured as follows. Section II describes the *EAM Pattern Catalog*, how it has been compiled, and its relationship to EAM patterns. Section III introduces exemplary EAM patterns for addressing architectural standardization, followed by a resume and outlook in Section IV.

II. The EA Management Pattern Catalog

The EAM Pattern Catalog is a collection of EAM patterns and has been created in two phases. Thereby, the focus lies on concerns, M-Patterns, V-Patterns, and I-Patterns, which are considered relevant and useful by experienced practitioners or are supported by literature. The catalog utilizes a consistent terminology and information organization to simplify the selection, adaptation, and integration of patterns.

The EAM Pattern Catalog is a structured approach, based on the experience of practitioners and on the concept of EAM patterns as building blocks of an EA management approach, which can be tailored to specific demands. Thereby, the whole approach focuses on addressing specific concerns and does not build an all embracing model that is ment be used for all thinkable concerns.

These building blocks avoid a giant, monolithic, and undocumented information model and replace it by a well documented and concern specific model, only including the

information needed by enterprise architects. This reduces the effort that has to be invested in collecting and maintaining the information to fill the repository.

Documenting typical EA management problems and their solution as EAM patterns is another advantage of the EAM Pattern Catalog, leading to an increasing amount of best practices available to public. This input can be used to incrementally improve the EAM Pattern Catalog. This improvement process is preferably implemented by a community caring about review cycles and periodical releases of the EAM Pattern Catalog.

In a first phase (October 2006 until July 2007), the EAM Pattern Catalog was initialized by our group based on input from the following sources:

- Research project Software Cartography, Technische Universität München, Chair for Informatics 19 (e. g. [7], [8], [22], [35])
- Partners of the research project Software Cartography
- EAM Tool Survey 2005 [28] and EAM Tool Survey 2008 [24]
- Enterprise Architecture at Work (ArchiMate) [18]
- Management von IT-Architekturen (Edition CIO) [10]
- IT-Unternehmensarchitektur, 2007 [19]

In a second phase (July 2007 until February 2008), the initial EAM Pattern Catalog was evaluated by 30 companies using an extensive online questionnaire to identify methodologies and viewpoints that are considered relevant and useful by practitioners[1].

A. Structure of the EAM Pattern Catalog

Based on the evaluation of the questionnaire results, the EAM Pattern Catalog in its present form covers[2]:

- 43 concerns (48 have been excluded),
- 20 M-Pattern (10 have been excluded),
- 53 V-Pattern (21 have been excluded), and
- 47 I-Pattern (19 have been excluded).

To support navigation and search, the EAM Pattern Catalog clusters concerns and methodologies according to the following EA management topics.

Technology Homogeneity describes methodologies analyzing and managing whether the application landscape relies on a homogeneous set of technologies and architectures.

Business Processes is concerned with analyzing the interaction of business applications, business processes, and related entities relevant to business at a high level of abstraction.

Application Landscape Planning is concerned with planning and analyzing the structure and evolution of the application landscape, focusing on current, planned, and target landscapes.

Support of Business Processes introduces methodologies for analyzing, how a specific business process is supported by IT.

Project Portfolio Management is concerned with managing the portfolio of projects changing the application landscape.

Infrastructure Management analyzes the technical infrastructure, on which the application landscape relies, and what impacts this infrastructure can have on the support that the business applications provide to the business processes.

Interface, Business Object, and Service Management summarizes methodologies concerned with analyzing and identifying services in the context of service oriented architectures (SOA). Thereby, the data flows created by communication via services, and the business objects exchanged via service interfaces are important aspects of the analyses.

B. Integrating EAM Patterns

Integrating the selected EAM patterns is an important aspect of using the EAM Pattern Catalog, or other sources of EAM patterns. Special attention has to be paid to potential conflicts, inconsistencies, or discrepancies, due to contradictory assumptions made by different EAM patterns, especially of different origins.

Such diverging assumptions may be completely valid for each of the EAM patterns itself, e. g. due to being based on different theories, being designed for different environments, or addressing different concerns. However, when simultaneously contained in one specific approach to EA management, diverging assumptions may easily turn out to be damaging or depriving results coming from the approach of their validity. This motivates the need to carefully manage such discrepancies in integrating EAM patterns, preferably avoiding them altogether. Thus, the below elaborations on integrating EAM patterns pay special attention to this issue.

1) Integrating M-Patterns: Selecting and integrating M-Pattern defines, how a specific set of methodologies interacts in order to address a given set of concerns.

This can be achieved by a *process model*, which provides the steps to be taken in EA management. Therein, it exhibits, according to [21], a basic characteristic of a method itself. Integrating M-Patterns can rely both on general research in the field of process models, from systems [21] or software engineering [30], and also specific process models for EA management, which are part of some EA management frameworks, e. g. the TOGAF ADM [34].

Basically, different reasons can lead to a methodology relying on specific assumptions:

- An M-Pattern may have been tested under specific conditions, other conditions can be known to be detrimental to the M-Pattern. An example may be factors known to benefit or impede effective knowledge management [9].
- An M-Pattern may be directly built on a specific (scientific) theory, which is valid under certain assumptions [27]. These assumptions then also have to hold for the M-Pattern to be applicable.

In order to be able to effectively integrate M-Patterns, these assumptions have to be documented with the respective pattern. This is necessary to enable a pattern integrator to manage inconsistencies when integrating M-Patterns. If e. g.

[1] See [6] for details of the selection process as well as relevance and usage statistics for each element.

[2] Numbers in brackets show the number of excluded EAM patterns and concerns. The complete list of EAM patterns included in the online survey has been consolidated to patterns identified to be relevant in practice.

one M-Pattern relies on information passing a formal review and publication process, while another M-Pattern wants to subject this information to wiki-style evolution, it has to be thoroughly checked whether and how these M-Pattern can be used together.

2) Integrating V-Patterns: Integrating V-Patterns may appear as the easiest integration task, as viewpoints are, according to the IEEE 1471 [17], supposed to be relatively self-contained. They are demanded to be able to address one or more concerns on their own, without demanding information from other viewpoints.

Adopting the idea of patterns to viewpoints offers the advantage to easily add layers to viewpoints. It is then possible to e. g. visualize applications on one layer and different key performance indicators on additional layers. This is the basis of the so called *layering principle* [12], which is borrowed from cartography (see Figure 2 for an example).

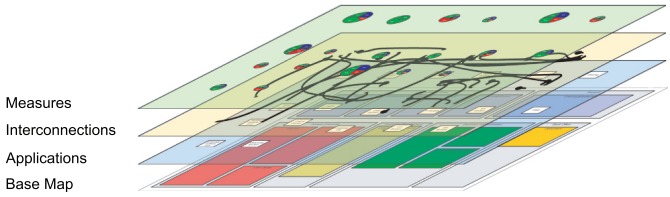

Fig. 2. Layering principle

3) Integrating I-Patterns: As described in [8], integrating I-Patterns strongly relies on the integrator's skills in conceptual modeling. It may e. g. be possible to identify identical classes within two I-Patterns to be integrated. However, this may not be as easy as it might seem at first sight, e. g. by identifying similar class names. Although EAM pattern designers should simplify pattern integration by naming different concepts clearly differently, also across different EAM patterns, this cannot basically be expected, especially, if patterns from various catalogs with distinct authors are to be integrated.

Issues in this respect may originate from the simplifications inherent in the creation of models [31], which may of course vary in different abstractions underlying different I-Patterns. A prominent example in this respect can be found in common abstractions of a *business application*. In some cases, a business application signifies an actual system installed in a specific environment, offering specific functionalities. In other cases, the term might specify the software itself, making no statements about specific installations. Usage of the term might also vary in respect to the versioning information included.

While, as stated above, sensible naming schemes, e. g. *BusinessApplication*, *DeployedBusinessApplication*, *BusinessApplicationVersion*, can help preventing such problems, one must not rely on this alone. Exact definitions of the used concepts have to be provided by the pattern designers, in order to enable the pattern integrator to find possibly contradictory definitions of concepts.

Additional integration problems are possible, of which one is subsequently exemplified. Consider a case where different I-Patterns (*I-x* and *I-y*) are integrated, of which one (e. g. *I-x*) employs a mandatory, e. g. *1..**, relationship to class contained in both patterns. In this case, it might be necessary to relax this relationship to an optional (*0..**) one, because otherwise collecting data for *I-y* might cause constraint violations in the model. Nevertheless, it should be possible to use *I-y* independently of *I-x*.

C. Usage scenarios of the EAM Pattern Catalog

Three different usage scenarios are supported by the EAM Pattern Catalog. The first one is to *Establish an organization-specific EA management approach via EAM Pattern Integration*. In this use case, it is assumed that EA management is introduced in a green field approach. In this case, first of all the pain points of the company, the so called concerns, have to be identified. This is supported by the list of concerns included in the EAM Pattern Catalog.

The selected concerns include references to M-Patterns that can be used to address these concerns. According to the approach sketched in Section I, the methodology described in the M-Pattern uses certain V-Patterns for visualizing aspects of the EA, which are referenced by the M-Pattern. Based on the selected V-Patterns, the associated I-Patterns have to be selected. The final step is to integrate the EAM patterns to an organization-specific approach for EA management.

The second usage scenario for the EAM Pattern Catalog is to use it to *Inspire and Asses an already implemented EA management approach*. This offers the possibility to compare the own EA management approach with best practices in use elsewhere. The EAM Pattern Catalog can e. g. be used to look for typical concerns, which occur in other companies. Additionally, the EAM Pattern Catalog can suggest visualizations that can be found in academia and practice, which may be helpful in an already selected EA management approach.

In addition to the application of the EAM Pattern Catalog in practice, there is a third approach to use the *EAM Pattern Catalog as a basis for academic research*. Currently, there is no common ground for research on EA management, meaning that there is no approach for EA management, which may be iteratively enhanced and extended. There are punctiform approaches for specific EA management topics, see e.g. [3] and [37], but these lack the integration into a holistic EA management approach, and the acceptance in wider communities.

The pattern based approach addresses this deficiency as it offers the possibility to improve single EAM patterns without having to create a completely new approach. Furthermore, the existing EAM Pattern Catalog can easily be extended due to the openness of the pattern approach.

III. EAM Patterns for addressing Architectural Standardization

Due to the relative novelty of the field of EA management, a company is likely to find itself in a situation, where its current application landscape is the result of an unguided evolution over a long period of time. Therefore, it often forms

a grown structure that does naturally not need to constitute the optimum, e. g. in terms of maintenance costs, flexibility, or the currently needed business support.

Put in other words, a number of different undesirable consequences is likely to arise from a lack of guidance in application landscape evolution. From these consequences, we chose the problem of low technological homogeneity and architectural standardization, for which we subsequently exemplify an approach based on EAM patterns [3] as introduced above. The concerns (see C–2 and C–19 in [6]) addressed by this approach can be stated as questions:

- Which business applications in the application landscape are individually developed applications?
- To which extent could a reorganization of the application landscape in respect to the used technologies affect e. g. licensing or maintenance costs?
- Which technologies used in the application landscape should be replaced, which should be kept?
- Where in the application landscape are architectural solutions used, and are there areas in the landscape, where those guidelines are breached?
- What is the build-up of an architectural solution?

In the above questions, a set of terms is introduced, which is subsequently clarified. The term *Technology* stands synonymously for a broad variety of concepts, ranging from programming languages as well as middleware platforms, over operating systems, to database management systems. All these concepts, while being very different concerning their usage, are considered similar here, as they all require a certain set of skills from the people which have to handle them. Resulting from this, the necessity to maintain a plethora of different skills can be considered as one of the undesired consequences of low technological homogeneity. These skills are e. g. necessary in development or maintenance.

The second important term relates to the dichotomy of *Standard vs. Individual Software*. While the first *BusinessApplicationType* refers to software products, which have been introduced into the company's application landscape after a customization process, the business applications of the second type are the results of development projects individually executed on behalf of the using company. Extensive usage of individual software may exert a multitude of undesirable effects, e. g. concerning maintenance. While maintenance for standard software is provided by the vendor and may be available in a maintenance agreement at little or perhaps no cost at all, maintaining individually developed software commonly requires a new development project. Therefore, maintenance might fall for all the risks, which are connected to executing a software development project.

Obviously, not all technologies as introduced above are likely to be used together, e. g. due to reasons of compatibility or integration. Further, information on how to integrate certain technologies has to be taken into consideration, when developing, evolving, or maintaining business applications. Therefore, many companies (see e. g. [5]) organize their knowledge on technologies suiting well on a higher level of abstraction, namely via *Architectural Solutions*. These solutions, sometimes also called *Technology Stacks* or *Platforms*, define a set of matching technologies, which are used in application architecture, e. g, on different tiers together with additional information on how to integrate these technologies into a solution to realize e. g. a four tiered architecture. To provide an example, one could think of an architectural solution "Internet Explorer 7.0, Tomcat 5.2, and Oracle 9.2i".

After having introduced the concept of the architectural solution, it is possible to explicate the fundamental issue of technological homogeneity not on the level of single technologies, but on the level of standardized technology bundles. It therefore could read as follows: Growing complexity is induced in the application landscape by the uncontrolled increase of used technologies. This can also be interpreted as an issue related to the usage of many different architectural solutions, even ones, which have not been defined to be intended according to corporate IT standards. Therefore, an analysis of the solution conformity of the application landscape can answer questions as the ones introduced in the concern. Subsequently, we provide methodologies appropriate for addressing these concerns.

A. Methodology

In addressing above concerns, an overview of the (deployed) business applications has to be given, especially taking into account the application's architecture and its corresponding technologies.

M-Pattern M–2 (see [6]) uses information on the software architecture used by business applications, which can be collected by the employees operating the actual applications, although it might be necessary to involve the developers of the applications in the collection process, as they are likely to have architectural information an ordinary user might not have.

Moreover, the information gathered has to be verified. For doing this, different possibilities can apply, ranging from automated plausibility and consistency checks to manual reviews. In this context both, actual users and developers, might have to be involved, as errors may occur in the data on different levels. The final step in the methodology, the creation of visualizations of the information according to the viewpoints presented below may also be helpful in verifying the data gathered.

Based on the information, a basic visualization of the business applications in the enterprise, e. g. according to their hosting at different organizational units is to be created. Having chosen this basic visualization[4] from [6], V–67 provides means to augment the visualization with additional information on

[3] Due to reasons of brevity the patterns are subsequently presented in a more colloquial form, not explicitly giving the problem, solution, and consequence section.

[4] We have chosen V–23, displaying this *hosting* relationship via clustering – a so called *cluster map* visualization.

standard conformance. In analyzing such annotated visualization, the focus is likely to be on the business applications not conforming to the respective architectural standard. On the one hand, such business applications might be looked at specifically, considering e. g.:

- Does it require not to conform with the standard?
- How much are costs thus induced? Who bears these costs?
- Has the wrong standard been prescribed for the business application?

On the other hand, analyses can also focus on the totality of the non-conforming business applications, e. g. looking at:

- What do they have in common?
- Are the standards inadequate for important parts of the application landscape?
- Are there organizational units for which there are no means of enforcing the standards?

Considerations, as the ones elaborated above, are inevitable connected to architectural standards being understood as a *boundary object*[5] between the community of the *enterprise architects* on the one and the community of the *software architects* on the other hand. A common understanding of these concepts is thereby necessary, to rise technological homogeneity while not overly restricting flexibility in software development.

M-Pattern M–4, based on M–2, is to be applied, in order to manage the evolution of the application landscape, especially in cases where high technological inhomogeneity is experienced. The methodology helps to control the evolution by setting architectural standards, i. e. defining architectural solutions and assigning them to business applications. In performing this, three major steps should be executed: *Setting Standards*, *Applying Standards*, and *Enforcing Standards*.

For setting standards, questions as the following should be considered, to determine the build-up of the respective architectural solutions:

- Which components may a business application consist of and how may these communicate?
- Which infrastructure software does a component rely on?

The next step, applying standards, is concerned with defining, which application should conform to which standard. In this context, it might be possible to breach the standard conformity, if beneficial for business success. However, if such an exception to the standard is allowed, it should further be considered, who (i. e. which business entity) receives the benefit of this exception, in contrast to who bears the cost induced thereby.

The final step, enforcing standards, involves methods for determining business applications not conforming to the standard (e. g. via the analyses as introduced in M–2), and evaluating the reasons for the business applications not to conform. If such an evaluation was performed, subsequently different proposals on future states of the application landscape and its conformance to architectural standards should be created and analyzed via the techniques as introduced with M-Pattern M–2. In discussing business applications currently not conforming to the standards, questions as the following should be considered:

- Has the wrong standard been set for a business application?
- Are there excessive costs for getting conformant to the standard?
- Is there a benefit from conforming to the standard, which cannot be realized in the context of the specific business application?
- Should a special charge for allowing a non standard solution be applied?

The methodologies, as alluded to above, are seamingly tightly interconnected, i. e. integrating them should be easy. This is true, as especially in executing the concrete analyses, the methodologies can be seen as complementary. In these discussions, visualizations showing the central aspects considered by the respective methodologies are likely to be used synoptically. Thereby, additional insight on interdependencies between non-conformity of business applications and technology related causes therefore might be gained. Here, especially the usage of M–2 techniques in evaluating application landscape proposals created in M–4 processes has to be noted.

B. Viewpoints

In order to address the concerns and answer the questions as stated above, the viewpoint presented in Figure 3 can be utilized.

V-Pattern V–67 is a layer for a *cluster map (V–Pattern V–24)* (see [6]). The cluster map is a software map utilizing nesting of symbols to show the grouping of business applications to logical units. Figure 3, for instance, visualizes which business application is hosted by which organizational unit by nesting the graphical representations of a business application into the symbol representing the corresponding organizational unit. The nesting of the rectangle *Online Shop (100)* inside the rectangle *Munich* thus means, that this organizational unit hosts the business application. It has to be noted, that the *cluster map* does neither specify how the rectangles representing logical units should be arranged on the map nor how the different elements nested into a logical unit should be positioned in relation to each other. Therefore, the positioning of the rectangles can be chosen to suite aspects as e. g. area minimization.

The viewpoint, of which an example is shown in Figure 3, further employs an additional layer, which illustrates via color-coding, which business application conforms to an architectural standard. Furthermore, approved exceptions to the defined standards are shown utilizing a symbolic annotation (check mark). Reasons for an exception to the defined standard might e. g. be the usage of a standard software or an interim solution, which is going to be changed in the near future.

[5]A boundary object is an object which allows members of different communities to build a shared understanding in respect to certain things. Boundary objects are interpreted differently by the different communities, and realizing as well as discussing these differences can lead to a shared understanding [32], [33].

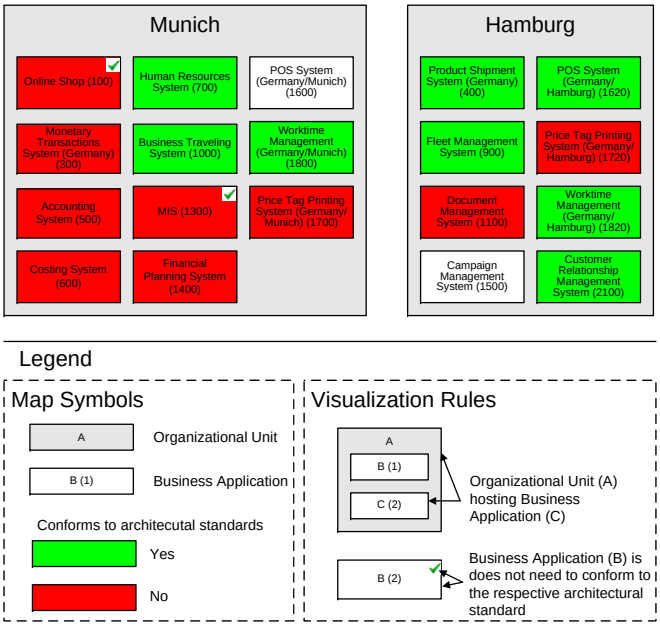

Fig. 3. V–67 on V–24: Example of the viewpoint indicating business application standard conformity

In order to integrate viewpoints *V–24* and *V–67*, the layering principle as introduced in Section II-B is utilized. Thereby, a colored rectangle of the same size is positioned above the rectangle representing the respective business application, if such information is available. Further, a check mark is placed in the upper right corner, if the business application is based on an approved exception.

Whereas *V-Pattern V–67* can be utilized to answer some of the questions in Section III-A, additional information about the technological build-up of an architectural solution is needed to answer questions on the structure of an architectural solution.

V-Pattern V–23 can be used to illustrate the used technologies for a given architectural solution. Figure 4 shows an exemplary view according to viewpoint V–23, which lists the technologies used by different architectural solutions. Thereby, the x-axis consists of the used technologies and the architectural solutions under consideration make up the y-axis. A cell is consequently marked, if the respective architectural solution uses the corresponding technology.

		Used Technologies						
		Apache 2.0.53	Bea Weblogic	DB2 6.0	IE 6.0	Oracle 9i	Proprietary Fat-Client	Tomcat 5.1
Architectural Solutions	ArchSol1a	x			x	x		x
	ArchSol1b	x	x		x	x		
	ArchSol2a			x			x	
	ArchSol2b					x	x	
	ArchSol3		x			x		x

Fig. 4. V–23 (v1.1): Technologies used by Architectural Solutions

In order to address the concerns as mentioned in Section III-A and answer the questions outlined in Section III, the viewpoints as introduced above can be utilized. Thereby, different information has to be gathered to create the viewpoints as illustrated above. The following section supplies the concepts on which the viewpoints are built and introduces an approach to integrate them.

C. Information Model Patterns (I-Pattern)

For every viewpoint as presented in the preceding section, specific information on the business applications, their underlying architectural solutions, and the supporting technologies has to be gathered. In order to complement this, we present the information model patterns needed in this context in a first step. In a second step, we elaborate on issues arising in integrating these I-Patterns and finally provide an integrated information model.

I-Pattern I–23 This information model pattern (see Figure 7) is taken from [6] and can be utilized to model data on the technologies used by architectural solutions.

Fig. 5. UML class diagram of pattern I–23

The concepts presented form the basis of a language usable for describing application architectures, i. e. they might be seen as elements of an *Architecture Description Language (ADL)*. In the context of this I-Pattern, the necessity to expand the model to a *full-scale* ADL is not given. The pattern is only the cutout of a more sophisticated model presented as I–66 in [6], which can be used for further detailing the architectural build-up of business applications.

The core concepts for describing the architecture, which are used in the context of I–23, can be defined as follows:

- An *ArchitecturalSolution* is a concrete stack of corresponding technologies, which are intended to be used together in realizing business applications, together with additional information on how to integrate these technologies into an complex architecture. Combining technologies together to a solution among others indicates, that components created from the technologies are technically suited for interaction and integration.
- A *Technology* represents a technical constituent of a business application, ranging from an implementation framework or platform to a database management system or user interface toolkit. Exemplarily spoken, technologies may be ”Apache 2.0.53” or ”Oracle 9.2i”.
- The association *uses* indicates, which architectural solution uses which technologies.

I-Pattern I–67 Below, we present version 1.1 of the information model pattern I–67 (see Figure 6).

The classes, as used in this pattern can be defined as follows:

- A *BusinessApplication* is a software system, which is part of an information system within an organization. An information system is therein according to [20] understood

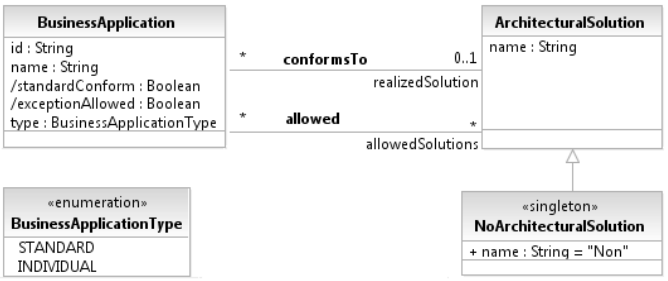

Fig. 6. UML class diagram of pattern I–67

as a socio-technological system composed of a software system (i. e. the business application), an infrastructure, and a social component, namely the employees working with the system. An information system is further described as contributing to the business process support demanded by the organization.

- The *NoArchitecturalSolution* represents the *Non-Solution*, i. e. it means, that an associated business application does not follow or does not need to follow any architectural solution.
- The *BusinessApplicationType* is used to model, whether a business application has been developed as a piece of individual software or is a bought standard solution.
- The association *allowed* explicates, which architectural solutions are per standard available for realizing the corresponding business application. Therein, the non-solution, as reflected by the singleton instance of the class *NoArchitecturalSolution*, is used to represent, that a business application does not need to conform to any architectural solution. This is especially necessary, to distinguish between the prescription of no solution vs. the absence of a prescription of that kind, i.e. missing data.
- The association *conformsTo* indicates, in accordance to which architectural solution a business application is actually realized. Such a solution might be the singleton instance of the *NoArchitecturalSolution*, thereby indicating, that no standard solution has been used. Further, no such information might be present, described by the absence of an associated solution.

For determining information about the standard conformance of the business applications, such as displayed in Figure 3, the derived attributes *standardConform* and *exceptionAllowed* are used. The values of these attributes are derived by expressions similar to the following[6]:

$$standardConform = \begin{cases} null & \text{for } (realizedSolution = null) \vee \\ & \quad (allowedSolutions = null) \\ true & \text{for } realizedSolution \in allowedSolutions \\ false & \text{for } realizedSolution \notin allowedSolutions \end{cases}$$

[6]These expressions might be realized in the Object Constraint Language (OCL). For reasons of readability, we chose a mathematical notation instead.

respectively

$$exceptionAllowed = \begin{cases} null & \text{for } allowedSolutions = null \\ true & \text{for } NoArchitecturalSolution \in \\ & \quad allowedSolutions \\ false & \text{for } NoArchitecturalSolution \notin \\ & \quad allowedSolutions \end{cases}$$

In deriving these values, the result *null* is used to indicate, that based on the current information no valid statements on the respective property can be made.

I-Pattern I–24 This pattern (see Figure 7), commonly alluded to as *cluster map information model pattern*, provides the basic concepts necessary for displaying business applications clustered according to the organizational unit, they are hosted at.

Fig. 7. UML class diagram of pattern I–24

This pattern introduces two additional concepts:

- An *OrganizationalUnit* represents a subdivision of the organization according to its internal structure. Possible examples are the entities showing up in an organigram.
- The association *hosts* indicates, which organizational unit is responsible for hosting a business application.

Integrating the Information Model Pattern As outlined in Section II-B, a naive approach to integration could be, speaking graphically, described as *glueing* these pattern together at common classes. In the exemplary patterns presented here, the classes *BusinessApplication* and *ArchitecturalSolution* would provide a good starting point. Nevertheless, such an integration approach might negatively influence the usability of the parts of the integrated information model – as e. g. multiplicity constraints imposed in one pattern might prevent a class common to two patterns from being instantiated without the need to provide further, but not instantly necessary information. The integrated information model as presented in Figure 8 takes this fact into account and performs suitable adaptations. These adaptations, that discriminate the information model from a purely *glued* solution, are subsequently detailed.

We relaxed the multiplicity constraint on the *Technology* end of the *uses* association connecting architectural solutions and their used technologies. This was performed, to accommodate to the existence of the singleton instance of the *NoArchitecturalSolution*, which does not need to specify technology usage. The second constraint to be relaxed applies at the *OrganizationalUnit* end of the *hosts* association between organizational units and business applications. Here, the end has been marked optional, such that a business application and its related architectural solutions can be modeled without first

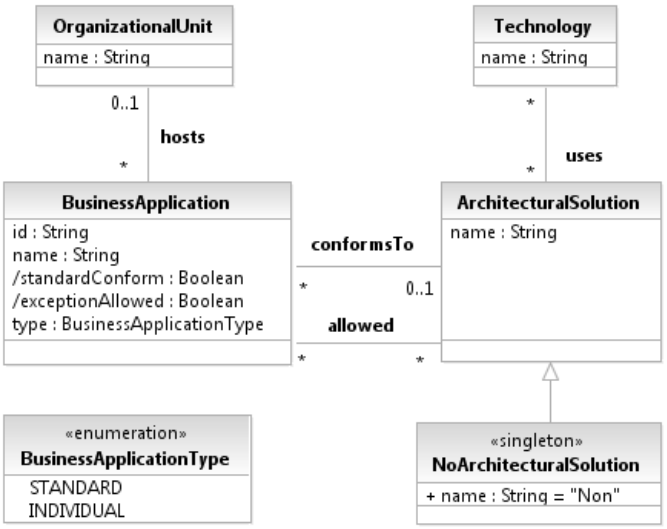

Fig. 8. UML class diagram of the integrated model

having to specify the organizational unit, the application is hosted at.

IV. Resume and Outlook

The article outlined an approach for consolidating, publishing, and advancing detailed practices for addressing EA management concerns. Thereby, the idea of design pattern, as originally introduced by [1], and made prominent in software engineering by [14], constitutes the foundation of the approach.

A research direction we pursue in this respect concerns the integration of EAM patterns. We are currently applying the EAM Pattern Catalog and parts thereof in EA management projects in practice, in order to validate our integration ideas presented in Section II-B, and find other challenges connected to integrating and implementing the patterns in a practical setting [2], [11].

The article also presented the *EAM Pattern Catalog*, a collection of best practices for EAM building on the pattern based approach. As another research direction for pattern integration, we are presenting the patterns in the catalog as extensions to EA management frameworks. As mentioned in Section I, frameworks like TOGAF or Zachman leave details about procedures to be taken, visualizations to be used, and data to be collected relatively open. Therefore, we view integrating the patterns from the catalog into an EA management framework as an effort complementing the high-level guidance provided by the frameworks with the detailed practices of the catalog.

As the catalog's utility depends on fostering, advancing, and establishing the specific patterns as accepted solutions to EA management problems, we are planning to evolve them via a community process. Thereby, we would like to apply approaches used in pattern communities in software engineering.

We are planning to put a web platform at the center of our activities to create an EA management pattern community, allowing users to exchange related information:

- Accessing the patterns, navigating along the pattern graph, and downloading related content, as e. g. the information models in XMI format, or related tools, e. g. for creating the visualizations
- Creating new versions of pattern, including the respective versioning workflows and statuses (in work, in review, etc.)
- Commenting on patterns, discussing about them, voting on patterns, or conducting surveys about them
- Reports of usage in practice

In creating an EA management pattern community, we are not planning to rely on a web platform alone. By initiating activities, from driving publications about worthy enhancements to patterns, to organizing workshops targeted at improving patterns in a cooperation of practitioners with researches, we plan to bring and keep alive a pattern community enhancing and establishing the pattern catalog.

To exemplify a possible enhancement, a potential approach for advancing patterns addressing architectural standardization could be introducing quantitative techniques, as e. g.:

- Metrics measuring degrees of architectural standardization, in order to be able to distinguish finer degrees of conformance, or
- Quantitative models for estimating the effect of architectural standardization or its absence on key quality attributes of an application landscape, as e. g. flexibility to change or incorporate new functionality, or scalability in cases of increased load. Such quantitative models might e. g. rely on simulations of processes running in and affecting an application landscape.

Techniques as mentioned above can be used in M-Patterns to make more founded decisions regarding which architectural standards to prescribe, and when to allow which deviations from them. By being able to measure the deviations, and predict their negative effects, checking whether they are worth a benefit expected from a nonstandard architecture becomes basically a business case decision.

As far as those approaches might seem from current practices, as e. g. put forward by EA management tools, or EA management frameworks, they point in a direction where EA management is less an art practiced by IT staff, but a science providing documented, predictable approaches to experts supporting business with the IT it needs. Building the EA management pattern catalog constitutes a key advancement in this direction to us.

References

[1] C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, USA, 1977.

[2] M. Böhme. *Softwarekartographie – Analyse und graphische Visualisierung von Teilen der Anwendungslandschaft des Klinikums der Universität München.* Bachelor thesis, Technische Universität München, Fakultät für Informatik, 2008.

[3] B. Braatz, C. Brandt, T. Engel, F. Hermann, and H. Ehrig. *An Approach using formally well-founded Domain Languages for secure coarse-grained IT System Modelling in a real-world Banking Scenario.* In 18th Australasian Conference on Information Systems (ACIS'07), 2007.

[4] C. Braun, R. Winter *A Comprehensive Enterprise Architecture Metamodel and Its Implementation Using a Metamodeling Platform.* EMISA 2005, pp. 64 - 79.

[5] K. Brendebach. *Integrierte Modelle und Sichten für das IT-Management.* Diploma thesis, Technische Universität München, Fakultät für Informatik, 2005.

[6] S. Buckl, A. Ernst, J. Lankes, and F. Matthes. *Enterprise Architecture Management Pattern Catalog (version 1.0, february 2008).* Technical report, Chair for Informatics 19, Technische Universität München, 2008, http://www.systemcartography.info/eampc.

[7] S. Buckl, A. Ernst, J. Lankes, F. Matthes, C.M. Schweda, and A. Wittenburg. *Generating Visualizations of Enterprise Architectures using Model Transformation (extended version).* Enterprise Modelling and Information Systems Architectures - An International Journal, Vol. 2(2), 2007.

[8] S. Buckl, A. Ernst, J. Lankes, K. Schneider, and C.M. Schweda. *A Pattern based Approach for Constructing Enterprise Architecture Management Information Models.* In A. Oberweis, C. Weinhardt, H. Gimpel, A. Koschmider, V. Pankratius, and Schnizler, editors, Wirtschaftsinformatik 2007, pages 145–162, Karlsruhe, Germany, 2007. Universitätsverlag Karlsruhe.

[9] T. Davenport, D. De Long, and M. Beers. *Successful Knowledge Management Projects.* Sloan Management Review, Vol. 39(2), 1998.

[10] G. Dern. *Management von IT-Architekturen (Edition CIO)*. Vieweg, Wiesbaden, 2006.

[11] T. Dierl. *Modelle, Methoden und Sichten für das Compliance Management.* Bachelor thesis, Technische Universität München, Fakultät für Informatik, 2008.

[12] A. Ernst, J. Lankes, C.M. Schweda, and A. Wittenburg. *Using model Transformation for Generating Visualizations from Repository Contents - an Application to Software Cartography.* Technical report, Technische Universität München, Chair for Informatics 19 (sebis), Munich, 2006.

[13] U. Frank. *Multi-Perspective Enterprise Modeling (MEMO) - Conceptual Framework and Modeling Languages.* In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences 35 (2002)

[14] E. Gamma, R. Helm, J.R., and J.M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley Professional Computing Series)*. Addison-Wesley Professional, 1994.

[15] N. Harrison, P. Avgeriou, and U. Zdun. *Architecture Patterns as Mechanisms for Capturing Architectural Decisions.* IEEE Software (September/October 2007), 2007.

[16] N. B. Harrison and P. Avgeriou. *Leveraging Architecture Patterns to satisfy Quality Attributes.* In: Proceedings of the ECSA 2007, LNCS 4758. Springer-Verlag Berlin Heidelberg, 2007.

[17] IEEE. *IEEE Std 1471-2000 Recommended Practice for Architectural Description of Software-intensive Systems.* IEEE Computer Society, 2000.

[18] H. Jonkers, L. Groenewegen, M. Bonsangue, and R. van Buuren. *A language for enterprise modelling.* In L. Marc., editor, Enterprise Architecture at Work. Springer, Berlin, Heidelberg, New York, 2005.

[19] W. Keller. *IT-Unternehmensarchitektur*. dpunkt.verlag, Heidelberg, 2007.

[20] H. Krcmar. *Informationsmanagement*. Springer, Berlin, 4 edition, 2005.

[21] K. Kronlöf, editor. *Method integration: Concepts and Case Studies*. John Wiley & Sons Ltd., Chichester, 1993.

[22] J. Lankes, F. Matthes, and A. Wittenburg. *Softwarekartographie: Systematische Darstellung von Anwendungslandschaften.* In: Wirtschaftsinformatik, 2005.

[23] J. Lankes and C.M. Schweda. *Using Metrics to Evaluate Failure Propagation and Failure Impacts in Application Landscapes.* In Multikonferenz Wirtschaftsinformatik. GITO-Verlag, Berlin, 2008.

[24] F. Matthes, S. Buckl, J. Leitel, and C.M. Schweda. *Enterprise Architecture Management Tool Survey 2008*. Technische Universität München, Chair for Informatics 19 (sebis), 2008.

[25] OMG: *UML 2.0 Superstructure Specification (ptc/04-10-02)*, 2004.

[26] OMG: *Meta Object Facility (MOF) 2.0 Core Specification (formal/06-01-01)*, 2006.

[27] M. Schermann, T. Böhmann, and H. Krcmar. *Fostering the Evaluation of Reference Models: Application and Extension of the Concept of IS Design Theories.* In 8. Internationale Tagung Wirtschaftsinformatik (to be published), Karlsruhe, 2007. Karlsruher Universitätsverlag.

[28] sebis. *Enterprise Architecture Management Tool Survey 2005*. Technische Universität München, Chair for Informatics 19 (sebis), 2005.

[29] Senate of the United States of America. *Public Company Accounting Reform And Investor Protection Act Of 2002.* In The Library of Congress - Congressional Record, Washington, 2002, http://thomas.loc.gov/beta/billView.jsp?&k2dockey=/prd/k2/congressional_record/xml/107/S15JY2/S15JY2-0013.xml@cong_record.

[30] I. Sommerville. *Software Engineering*. Pearson Education Ltd., Edinburgh, 7 edition, 2004.

[31] H. Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, Wien, 1973.

[32] S.L. Star and J.R. Griesemer. *Institutional Ecology: "translations" and coherence: Amateurs and professionals in berkeley's museum of vertebrate zoology.* Social Studies of Science, Vol. 19(3):387–420, 1989.

[33] J. Strübing. *Soziale Welten - Arenen - Grenzobjekte.* In 29. Kongress der Deutschen Gesellschaft für Soziologie (Sektion Wissenschafts- und Technikforschung), Freiburg, 1999.

[34] The Open Group. *TOGAF (The Open Group Architecture Framework). Version 8.1 "Enterprise Edition"*. The Open Group, 2003.

[35] A. Wittenburg. *Softwarekartographie: Modelle und Methoden zur systematischen Visualisierung von Anwendungslandschaften*. PhD thesis, Fakultät für Informatik, Technische Universität München, 2007.

[36] J. Zachman. *Extending and Formalising the Framework for Information Systems Architecture.* IBM Systems Journal, Vol. 31(3), 1992.

[37] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, and N. Schuster. *Reusable architectural decision models for enterprise application development.* In S. Overhage, C. Szyperski, editor, QOSA 2007, Springer, Heidelberg, 2007.