A Domain-Specific Language for EAM Entities and Views

- Final Report -

Guided Project Type B

Master’s Program Information Systems

Guided by Prof. Dr.-Ing. Stefan Bente

Winter Term 15/16

by Anass Amajoud, Muaaz Asif, Kieu Dai Tran, Markus Engelke, Stefan Kirsch, Jannik Kutscher, Dominic Leuchter, Regys Mene, Christoph Roehr
Abstract

This work provides a Domain-Specific Language (DSL) for commonly used EAM views and entities. It can be used in order to enable IT and non-IT stakeholders to address company related concerns by building views and entities easily and without requiring deep IT/EAM know-how. Exemplary, a proof of concept illustrates the usage of the groovy-based language. Besides describing views and entities, the DSL supports companies dealing with its strategic planning by supporting the development process of EAM metamodels.

Based on literature research and conducted personal interviews with EAM experts, catalogs with commonly used views and entities were populated. Based on this, commonalities of its characteristics were investigated. In favor of handling the voluminous catalogs, grouping similar artefacts and underline its main themes, a classification approach constitutes an important component of this project.
# Table of Contents

1. Introduction

2. Definition of terms

3. Business scenario

4. View Catalog/EAM Views
   4.1 Creation of View Catalog
   4.2 Classification
   4.3 Concluding remarks

5. Entity Catalog/EAM Entities
   5.1 Layers
   5.2 EAM Quality Entity
   5.3 EAM Entity Attributes
   5.4 Concluding Remarks

6. DSL Code Conception and Development
   6.1 Basic conception and derivation
   6.2 DSL code implementation
   6.3 Proof of Concept

7. Conclusion

References

Appendix
List of Abbreviations

BPMN .................................................................................................................. Business Process Model and Notation
DSL ...................................................................................................................... Domain-Specific Language
EAM ..................................................................................................................... Enterprise Architecture Management
ISO ...................................................................................................................... International Organization for Standardization
KPI ....................................................................................................................... Key Performance Indicator
NVA ..................................................................................................................... non-value adding activities
TOGAF .............................................................................................................. The Open Group Architecture Framework
TUM ...................................................................................................................... Technische Universität München
List of Figures

Figure 1: EAM metamodel, derived from ISO 42010..........................................................3
Figure 2 Separation of concerns; model, view and viewpoint..............................................12
Figure 3 Visualization of a View from EAM Pattern Catalog...........................................13
Figure 4 Visualization of a view from Marc Lankhorst......................................................13
Figure 5 Classification of EA viewpoints by Lankhorst....................................................14
Figure 6 Excerpt from View Catalog ................................................................................15
Figure 7 "Standard conformity layer", extracted from TUM Pattern Catalog .......................17
Figure 8 Metamodel for EAM views ..................................................................................18
Figure 9 Meta-model of concepts of EAM entity catalog ..................................................21
Figure 10 Metamodel of the scenario ................................................................................26
Figure 11 Example Draft of a view .....................................................................................28
Figure 12 Class Overview ................................................................................................29
Figure 13 Class Diagram View ..........................................................................................29
Figure 14 Class Diagram Group and Visualization Rule ...................................................30
Figure 15: Class Diagram Entity .......................................................................................31

List of Tables

Table 1 Sub-Goals for each Team.........................................................................................2
Table 2 Grouping of views..................................................................................................16
Table 3 Grouping of views - Themes..................................................................................17
Table 4 Grouping of views - Level of abstraction ...............................................................18
Table 5 Grouping of views - Theme: Process....................................................................19
Table 6 Grouping of views - Theme: Application/Technical .............................................19
Table 7 Grouping of views - Theme: Architecture.............................................................19
Table 8 Grouping of views - Theme: Organization.............................................................19
Table 9 Strategic Layer........................................................................................................23
Table 10 Business Layer ...................................................................................................23
Table 11 Application Layer ...............................................................................................24
Table 12 Technology Layer.................................................................................................24
Table 13 Quality Entity.......................................................................................................25
Table 14 Shared Attributes for all entities.........................................................................26
Table 15 Specific Attributes for each of entities................................................................27
1 Introduction

The close alignment between Business and IT is considered as one of the biggest concerns of companies nowadays.\(^1\) Enterprise Architecture Management (EAM) provides approaches to make clear the dependencies between different enterprise layers and components, such as Strategic components, Business processes and IT applications. The linkage between different architecture layers (e.g. Business, Application and Technology Layer)\(^2\) can be conducted by a metamodel approach which specifies consistency of various architecture artifacts on different layers and in different views. Actually this metamodel is helpful in order to build a domain specific language (DSL) which will provide a visual language that facilitates the definition of EAM elements, concepts and models.\(^3\)

Up to this point there is no commonly accepted standard DSL for EAM. DSL, being a programming language, is a helpful tool for different stakeholders which should enable them to address their concerns within a specific domain. This language could cover technical domains like (models, workflow, business processes) or functional domains (insurances, banking or healthcare).\(^4\)

In this project, it is aimed to develop a DSL in order to facilitate the understanding of EAM by describing EAM views and entities. Such a language will make the application of EAM to a specific business area more automatic and agile. The model representation, the definition and EAM analysis are presented at an abstract level with the DSL, which makes the design of the enterprise architecture very practical as well as more intuitive and even understandable for non-enterprise architects.

A business scenario is used to put a specific concept into context of a real environment. It serves the purpose to check if the proposed concept is working, usable and if it is able to meet business requirements in a scenario which is typical for the problem domain. A business scenario can help to identify and understand business needs and its results can be used to refine the concept accordingly, as well as serve as a practical example or proof of concept. A meaningful business scenario represents a significant business need or problem, and enables stakeholders to better understand the problem context, the solution to the problem and thus the value of the applied.

Furthermore, it is intended to develop a deep research on all the main concepts of EAM first. Thereby, a standard catalogue should contain most common entities and views, which served as building blocks for constructing a DSL. To check and support the research, four personal


\(^2\) Derived from the enterprise architecture modeling language ArchiMate (2.1), The Open Group

\(^3\) Buckl et al., “Enterprise Architecture Management Pattern Catalog.”

\(^4\) McLeod, “Cooking up a MEAL: Creating a Meta Enterprise Architecture Language.”
interviews with EAM experts where conducted. The insights, gained by these interviews will be added to the respective chapters. Based on researches and these interviews, common definitions of core concepts of EAM like ‘entity’, ‘meta-model’, ‘view’ and ‘viewpoint’ were investigated and used. Next step was to provide an Entity/View catalogue that contains the most common entities and views in EAM. Based on the two catalogues, the last step was the development of the specific DSL dedicated to EAM.

The resulting language serves as a standard gateway between the IT- and non-IT stakeholders and reduces development and design effort needed in EAM by providing a higher level standard abstractions for DSL. The document presents in details necessary steps for the development of the DSL which eases the application development in the domain of EAM. It also focuses on the results of this research and analysis in EAM which led to carry out entities and views catalogues, the document points out in the end the concepts of how to develop a DSL for EAM using groovy language which provides domain-specific constructs for entities and views within the pervasive EAM domain.

The project is organized in consecutive mini-goals organized in two themes: Entities and views.

<table>
<thead>
<tr>
<th>Team Entities</th>
<th>Team Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Identify the common Entities Based on research</td>
<td>• Identify the common views Based on research.</td>
</tr>
<tr>
<td>• Identify the common Entities Based on Interviews</td>
<td>• Identify the common views Based on Interviews</td>
</tr>
<tr>
<td>• Build the Entity Catalog (Attributes and relationships)</td>
<td>• Build the View Catalog (Visualization and Classification)</td>
</tr>
<tr>
<td>• Build DSL Classes with Attributes for Entities</td>
<td>• Build DSL Classes with Attributes for views</td>
</tr>
</tbody>
</table>

Table 1 Sub-Goals for each Team

The project is held by a team of nine members and conducted under the guidance of Prof. Dr. Stefan Bente. The Team members are divided into two sub-teams: ‘Team Entities’ is responsible for research and analysis of relevant EAM entities, building an entity catalog and specifying DSL classes with respective attributes for entities. The second team is ‘Team Views’ having as mission making research and analysis of EAM relevant views, building a view catalog and developing a DSL relevant to EAM views.
2 Definition of terms

The literature in the context of Enterprise Architecture Management provides a great variety regarding the definitions and common terms that are used. This chapter gives an overview over the core concepts that are used and dealt with during the project itself. It is important to state that some of the definitions were adapted in their original meaning, like the ones for viewpoint and view, where others are created by this project team to serve this project’s needs, like the definition for entities.

EAM metamodel

The core concept that connects the terms of this chapter with each other is an EAM metamodel, which is derived from the ISO 42010. With respect to the special needs for the development of a domain-specific language, the original model was reduced to the terms that are used in this paper and enlarged by terms that are essential.

As shown in Figure 1, grey elements are directly derived from ISO 42010, whereas yellow elements are adapted or adjusted with respect to the material that was used during the research. In this model, stakeholders can have one or more concerns, which are later addressed by views or framed by visualization rules. These visualization rules govern the view itself. Views always contain one or more entities. These terms and elements will be explained in more detail in the respective definitions. In conclusion, the model gives a broad overview of the terms and their connections within this paper.

---


6 Ibid.
According to the definitions of Rozanski and Woods, „A viewpoint is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views.” Consequently it can be stated, that a viewpoint in the context of this paper means the frame, from which a view can be created. These different frames are reusable for different kinds of views and can be seen as an interface between the stakeholder and the actual view. Viewpoints always cover one or more concerns.

EAM View

A view can be defined as “a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders.” In other words, a view represents the actual instance of a viewpoint, in which the actual defined model can be seen from. Regarding the field of EAM, it means that a view generally shows a part of the enterprise from a specific perspective which is defined by the stakeholder and his concern. For example, a stakeholder could be a chief architect with the concern of reducing non-value adding processes from the business processes. A typical view in that case could be a process map, which shows every element in a specific process. In conclusion, the used view gives this overview and can be classified as a support for the decisions the stakeholder makes.

Stakeholder & Concern

Derived from the ISO 42010, a stakeholder is “any person, organization or group with an interest in the system. [...] Within the standard, a stakeholder has one or more (architectural) concerns pertaining to the system of interest.” Thus said, a stakeholder is anyone with an intersection with the discussed system or environment. The goal-stakeholder matrix in the business scenario of this document lists a few typical examples of stakeholders. The actual concern of a stakeholder is “any interest in the system” and can be hold by one or more stakeholder.

EAM Entity

An EAM entity is an abstract and unique concept, which in the context of enterprise architecture may be a process, application, component etc. and defined by its own attributes. EAM entities may also have different relationships with other concepts too.

---

8 Ibid.
10 Ibid.
**Attribute**

An attribute defines a specific characteristic of an EAM entity or view. In these catalogs sets of attributes are defined for EAM entities or views. Furthermore, specific attributes for each EAM entity or view are defined in case of further drilldown into each of EAM entity or view.

**Domain Specific Language**

A domain specific language (DSL) is “a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.”[^1] This language is strongly related to a specific problem domain, which is in this case the enterprise architecture itself. Although the DSL is a formal language, one of the most important attributes is the easy use by people who have little or no knowledge of programming languages at all. More important is the knowledge of the specific domain the language is set in. This means that the language that is developed in this paper is for example usable by experts and users of EAM.

3 Business scenario

The following business scenario serves as a continual medium to illustrate the described concepts and their relationships in realistic case studies.

For this purpose, “CornerStone Care” as an insurance company will be introduced, which consistently grew over the last three decades. Currently, the company has six business areas with 42 main processes as well as 162 sub processes spread across 8 different regions. The IT department consists of 250 employees and has a budget of 500 million US Dollar per annum. The IT landscape grew organically over the course of 35 years and currently consists of 1100 different IT-systems which are between 1 and 35 years old.

The company is currently facing an increased number of customer complaints about errors in the automatic processes and long waiting times for the company services. The Management identified the heterogeneous IT-landscape with its multitude of different interfaces, different operating systems and programming languages as the main problem. Hence, the Chief Operating Officer hired a Chief Enterprise Architect and assembled a small task force to address this issue and start an enterprise architecture project. The task force consists of the Chief Operating Officer himself, who will have a keen eye on the business side of the project, with a special interest in lean processes, short term goals and issues regarding employees and productivity. The new Chief Enterprise Architect, who aims to align the strategic vision to information technology, and a process manager will give hands-on input about the operating layer of the affected workflow.

At the kickoff meeting the three team members agreed on the following measures to address the issues:

- Eliminate non-value adding activities (NVA) from business processes to speed-up the workflows and be more efficient
- Reduce number of different hardware/software systems to #-number to increase maintainability and reduce errors
- Reduce amount of self-developed IT systems to #-number to increase robustness
- Use the new lean and flexible architecture to increase self-service offerings to 50%

Based on this, a so-called Goals/Stakeholder-Matrix was derived. The purpose of the matrix is to have an overview about the maxims each stakeholder has about the respective concern, as well as to have a mapping between the corresponding entities and views which addresses the maxims.
<table>
<thead>
<tr>
<th>Stakeholders/goals</th>
<th>Chief architect</th>
<th>Chief Operating Officer</th>
<th>Process manager (responsible for birthday certificate process)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Views: 'IT should be used to increase the efficiency of the process (e.g. by automating some steps)' - V-12: Business Process and Business Function Relationship - V-17: Process Support Map - V-18: Service-based Business Process Support Map</td>
<td>Views: 'Process takes too much time' 'Too many resources (e.g. clerks) are needed during the process'</td>
<td>Views: 'Birthday certificate process has too much waiting time periods' - V-12: Business Process and Business Function Relationship - V-17: Process Support Map - V-18: Service-based Business Process Support Map</td>
</tr>
<tr>
<td></td>
<td>Views: 'It's hard to introduce new SW functionalities due to consideration of so many different SW/HW platforms.' - V-6: Clustering by Standard - V-8: Knowledge Needs - V-17: Process Support Map</td>
<td>Views: 'Employees have to get familiar with too many technologies' - V-6: Clustering by Standard - V-8: Knowledge Needs</td>
<td>Views: 'This process consists of too many different software' - V-17: Process Support Map</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| **Reduce amount of self-developed IT systems to ## to increase robustness** | Views:  
'Maintenance of self-developed SW is hard to achieve because of outdated programming platforms'  
'Many new SW applications cannot be integrated in self-developed systems'  
- V-6: Clustering by Standard  
- V-8: Knowledge Needs  
- V-17: Process Support Map | Views:  
'Outdated software doesn't handle the rising amount of data sufficient'  
- V-6: Clustering by Standard  
- V-8: Knowledge Needs | Views:  
'Our total process time of relatively simple processes (e.g. birthday certificate) is much to cost consuming compared to competitors'  
- V-17: Process Support Map |
| | Views:  
'What kind of interfaces do we have to develop for self-service offerings?'  
'How is data structure influenced by the self-service offerings?'  
- V-18: Service-based Business Process Support Map | Views:  
'How many working hours can be saved by introducing self-service offerings?'  
- V-18: Service-based Business Process Support Map | Views:  
'What kind of activities of this process are affected by introducing self-service offerings?'  
- V-18: Service-based Business Process Support Map |
Business scenario

The principle behind the matrix will be explained with the following example.

Example: In the first row, first segment, one can see the goal “Eliminate non-value adding activities (NVA) from business processes” and in the following segments the corresponding values for the specific stakeholders.

Chief Enterprise Architect: The interest of this stakeholder regards to the business side as well as the IT. So, a respective maxim regarding this concern could be “IT should be used to increase the efficiency of the process (e.g. by automating some steps)”. If you have determined this maxim, you can start to identify corresponding views. For this example, three views which address this maxim from the TUM-View-Catalog were identified:

- V-12: Business Process and Business Function Relationship
- V-17: Process Support Map
- V-18: Service-based Business Process Support Map

When one identifies the views, you can evaluate what entities are needed to build the views. For this example, these entities are:

- Business Function
- Business Event
- Business Process
- Business Application
- Organizational Unit
- Application Service

In the following documentation, this business scenario and the derived Goals/Stakeholder Matrix will be used continuously to illustrate the relevant concepts of this project.
4 View Catalog/EAM Views

In this project, the view catalog serves as a compilation of Enterprise Architecture views that encompasses a wide array of view instances that correspond to various organizational units, stakeholders and entities associated with an enterprise. The catalog is populated by two main sources. These sources are also responsible for some of the attributes, and structural organization in the view catalog. Additionally, using multiple sources enables to have an implementation approach backed up by previous research and tried and tested methodologies.

One benefit of using these sources is that they contain terms that are commonly used in business & IT. This reduces the chance of misconception for terms that are being used in the catalog since having a common understanding of EAM terms and definitions by business and IT stakeholders is one of the staple outcomes of this project. Ultimately the domain experts should be able to manage their own rules, associations, terminologies and configurations for views with the DSL.

In the process of creating a DSL, the idea of composing a View Catalog was conceived as a means to have a standardized source of DSL language structure and taxonomy. By maintaining a standardized method of entering views in the view catalog, the DSL language structure is conceived. In this context, the manifold purposes of a View Catalog are:

- To orchestrate an adequate structure for DSL views by creating required attributes for an enterprise view.
- To bridge the development gap between DSL concept and DSL implementation, since the artifacts from the view catalog are objectively used in the DSL. This implies that these views would be able to map on a generalized DSL.
- To align the work of Team Views with Team Entities by exploring and explicitly specifying the corresponding entities associated with each view in the catalog.

As defined in the previous chapter, viewpoint is what instantiates a view. It makes a view visible, meaningful and productive. Typically, a viewpoint contains an assortment of visualization rules, syntax, cues and techniques that make it possible for a view to be understood by its corresponding stakeholder or stakeholders. Of course this means that different stakeholders have different viewpoints for the same view. In other terms, different stakeholders may derive different meanings from the same artifacts. These visualization rules are included for each view in the view catalog so that there is a guide for comprehending each view in order to avoid miscommunication. However, it is important to point out that there is loose syntax for creating viewpoints and visualization can take various forms ranging from diagrams to tables, dynamic visualizations like short movies or even figures and cartoons. This loose syntax exists so that the corresponding stakeholder is able to use real-life artifacts to understand, create and communicate the view. Although this statement may seem like a paradox, lesser restrictions help achieve better standardization.
4.1 Creation of View Catalog

Now that there’s adequate explanation about views and viewpoints, the next step is to discuss the creative process of the view catalog. The process of creating the view catalog was naturally inspired by the reference literature. As previously mentioned that many artifacts in the catalog are derived from the sources, it would be ideal at this point to discuss the sources used to create, and populate this view catalog.

Sources

The first source is EAM Pattern Catalog created by Buckl et al. at the Technische Universität München. This source considers a holistic approach for cataloging Enterprise views for an Enterprise Architecture. It considers various Enterprise Architecture frameworks to come up with a structure for creating an Enterprise Architecture view. As such, it is fraught with views from real business cases and solidifies inferences and approaches for structure and content in the view catalog. The EAM Pattern Catalog identifies views on the basis of Enterprise Concerns. These concerns, also known as “maxims” from the Goals/Stakeholder Matrix are the driving force for the creation of EA views. In the source, they refer their approach as being similar to design patterns used by the software engineering community. This critical link to software engineering further strengthens the claim made by one of the interviewees that “Without Enterprise Architecture, IT becomes a commodity.”

Although the view catalog looks like a matrix, its structure is essentially similar to the one used by EAM Pattern Catalog. Where the Pattern Catalog represents all patterns one by one in detail, contrarily the View Catalog is represented in the form of a matrix because view attributes take the spotlight. This is due to the fact that the final work product of this project is to have a DSL for EAM views and Entities. This structural representation enables readability for deriving similarities and differences as well as drawing comparisons between the attributes of a view. The EAM Pattern Catalog is meant as a starting point for the pattern community, whereas the View Catalog is meant to be used as a tool for creation of the DSL. Another area where the EAM Pattern Catalog is lacking is that it does not consider EAM Entities isolated. Furthermore, the View Catalog provides the formalization and structure required for creating the DSL which is lacking in the EAM Pattern Catalog.

The second source is “Enterprise Architecture at Work” by Marc Lankhorst et al. which deals with enterprise architecture modelling by putting ArchiMate language at the front seat. Another important aspect of this source is that it strives for standardization of Enterprise Architecture by mentioning that ArchiMate 2.0 aligns with TOGAF. In the context of the View Catalog, this source provides knowledge about viewpoints and visualization. It provides the idea of mapping concerns to stakeholders and this is manifested in the view catalog by mapping views to stakeholders. Furthermore, it helps delineate the terms models, views and viewpoints which can be illustrated by the following figure.

---

12 Buckl et al., “Enterprise Architecture Management Pattern Catalog.”

13 Schwaiger, EAM Views and Entities.

14 Lankhorst, Enterprise Architecture at Work.
The correlation between View and Viewpoint as shown in Figure 2 exists similarly in the resulting DSL where viewpoint is a subclass of a view. This source also helps to identify the attributes for “Purpose” in the view catalog, which will be explained in detail in the coming chapter 4.2 Classification. Since the emphasis is on standardization, all of the views in this source are visualized according to ArchiMate standards. Finally, the views in this source also mention the corresponding entities which gives an idea about their relationships and helps create a link between both the View and Entity Catalog.

**Viewpoints and Visualization**

It has been already established that viewpoints give visualization cues to enterprise views. Although at first it may seem that viewpoints and the visualization techniques have little to do with the creation of a DSL, it is imperative to understand that a taxonomical approach to viewpoints and visualization, albeit at a meta-level, leads way to standardized creation of DSL. On the contrary, viewpoints play arguably a more important role in communicating architecture than views. As previously mentioned, the majority of information for viewpoints and visualization comes from Lankhorst, 2013.\(^{16}\) Viewpoints and visualization give context to EA views in the sense that they visualize a stakeholder and its concern to a business case.

In the view catalog, the visualization column textually describes how each view gets visualized. Visualization is slightly different between TUM and Lankhorst views. This can be more clearly explained by showing an example from each source. Figure 3 shows a visualization example from EAM Pattern Catalog where the visualization is supported by a legend that describes what each notation means. Most views in this source are supported by this legend as the visualization can be otherwise difficult to understand due to the variety of symbols being used. Since views in the EAM Pattern Catalog are not visualized using a standard modelling language, the types of visualizations vary more than they do in the second source.

---

\(^{15}\) Ibid.

\(^{16}\) Ibid.
Figure 3 Visualization of a View from EAM Pattern Catalog

Figure 4 shows the visualization of a view from the Marc Lankhorst source where different shapes, colors, symbols and links visualize different artifacts. In the DSL code all of these attributes are implemented in order to stay true to this approach. In the view catalog, all of these visualizations are described textually. These visualization types are implemented as “Visualization Rules” in the DSL. It should be pointed out that the views from the view catalog must be instantiated in the DSL. Otherwise, the DSL contains only classes, hierarchies, relationships and class attributes.

Figure 4 Visualization of a view from Marc Lankhorst

---


Association with Business Scenario

The integration of view catalog also factors into account the business scenario of CornerStore Care, an insurance company. This company will serve as test case for verification and validation of the language syntax that goes from the view catalog into the creation of the DSL. As previously mentioned, the DSL implementation exists as generalized implementation of classes. In the proof of concept, explained in the later chapters, one of the maxims from the Goal/Stakeholder Matrix in the business scenario will be mapped according to the attributes provided in the View and Entity Catalog. Each maxim in the Goal/Stakeholder matrix corresponds to a set of views which exist in the View Catalog. These views are different for different types of stakeholders depending on the level of detail required to be visualized.

Choosing Attributes

The selection of attributes for creation of the view catalog was an extensive, thorough process. Although most of the attributes were derived from reference literature, it was important to create and assign unique attributes to the views because of a couple of reasons:

- The reference literature does not aim to help create a DSL-type syntax.
- Both sources are not considering EA entities at the same conceptual level
- Stakeholders are not explicitly mentioned as part of EA views in the reference literature
- Classification of views is another important topic which had disparate semantics in both sources

Figure 5 Classification of EA viewpoints by Lankhorst

In this chapter, attributes chosen for the view catalog will be explained with adequate reasoning about their inclusion, purpose and usage in the DSL.

---

Lankhorst, *Enterprise Architecture at Work.*
The attributes are as follows:

I. **Version**: Keeps track of the version of an EA view. Currently, all views are in their original version.

II. **Purpose**: This attribute is derived from the Marc Lankhorst source and its values can be seen in Figure 5. *Deciding* is used in decision making, *Designing* is used for navigation and design support, and *Informing* is used to explain, convince and obtain commitment.

III. **Group**: This attribute is used to classify viewpoints and is explained in detail along with the classification process in the concurrent chapter 4.2 Classification.

IV. **Description**: Provides a description of each EA view in the form of a brief summary by explaining its purpose.

V. **Visualization**: As explained previously in some detail, visualization column in the view catalog provides a textual description of what the viewpoint visualization looks like.

VI. **Source**: This column shows which source the corresponding EA view belongs to.

VII. **Main entities**: In order to stay close to the Entity Catalog as well as to aid in the DSL development, the main entities column enlists all the entities associated with each EA view.

VIII. **No. of Entities**: This column gives a total number of entities associated with an EA view.

IX. **Stakeholder**: Last but not the least, the stakeholder column gives the stakeholders associated with each view.

---

20 The full catalog can be found in Appendix B Tables - View Catalog, p. 46
4.2 Classification

Populating the view catalog led to a voluminous document, containing many similar views and losing its clear outline. To ensure a coherent overview it needs to be simplified. This issue is addressed by the classification or grouping of the views. Furthermore, the classification may not only simplify the current structure of the catalog, but also reduce the difficulty of adding views later on. Therefore, one of the most important goals regarding the classification is to keep the total number of different groups as small as possible.

As a first step, all identified attributes are checked regarding their ability to distinguish views from one another. For example, an attribute like “Purpose” does not represent a good choice at this point, because most of the instances serve the purpose of “informing” - at least to some extent - which increases the difficulty of distinguishing views.

The other attribute based on Lankhorst, “Level of abstraction”, appears to be a much better fit. In contrary to “Purpose” its values (“Detail”, “Overview” and “Coherence”) are more evenly distributed through the analyzed views, qualifying the attribute as a part of the classification. But, since there is little benefit in a distinction based on just one criterion, the approach to group views should be based on different adjuncts.

Following the above described procedure the grouping of views is composed of the further listed attributes (and its possible values):

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Leading question</th>
<th>Possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theme</td>
<td>What is the view about?</td>
<td>Application/Technical</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Process</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Organizational</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Architectural</td>
</tr>
<tr>
<td>Level of abstraction</td>
<td>How detailed is the view? Does it consist of different levels of abstraction?</td>
<td>Overview</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Coherence</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Detail</td>
</tr>
</tbody>
</table>

Table 2 Grouping of views

As discussed before, one of the main goals of the classification is to create as few groups as possible while assuring the possibility to assign each view to exactly one group. Therefore, only two attributes are selected:

**Theme**

Dealing with the topic of a view, “Theme” is deducted from both the name and the content of a view. While also considering a views environment or context, its “main statement” is the focus of this attribute. Leading questions are: What is the view mainly about? What is the core message? What kind of topic is addressed?
Taking into account the views included in the catalog, the following themes are identified:

<table>
<thead>
<tr>
<th>Theme</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application/Technical</td>
<td>Versioned software-based platform to solve specific problem(s)/ Hardware-based platform to provide resources.</td>
</tr>
<tr>
<td>Process</td>
<td>A sequence of activities with specific input and output.</td>
</tr>
<tr>
<td>Organizational</td>
<td>Display of competences and hierarchy within a certain environment.</td>
</tr>
<tr>
<td>Architectural</td>
<td>“Fundamental concepts or properties of a system in its environment, embodied in its elements, relationships, and in the principles of its design and evolution.”</td>
</tr>
</tbody>
</table>

Table 3 Grouping of views - Themes

The example below illustrates the fine line between topics covered in a view and its “main statement”. Although there are subsidiaries with their respective business application used depicted, the view is not about the organization of the enterprise, but the applications and their conformity to architectural standards. Taking into account the name of the view (“Standard Conformity Layer”) it becomes obvious that it should be assigned to the group of “application/technical” views.

![Diagram of view types]

Figure 7 "Standard conformity layer", extracted from TUM Pattern Catalog

---


**Level of abstraction**

“Level of abstraction” is an adjunct adopted from Lankhorst, who refers to it as the “content” of a view and defines the different levels of abstraction as listed beneath:

<table>
<thead>
<tr>
<th>Level of abstraction</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Details</td>
<td>“Views of the detailed level typically consider one layer and one aspect […] Examples of views are a BPMN process diagram and a UML class diagram.”</td>
</tr>
<tr>
<td>Coherence</td>
<td>“At the coherence abstraction level, multiple layers or multiple aspects are spanned. Extending the view to more than one layer or aspect enables the stakeholder to focus on architecture relations like process–use–system (multiple layer) or application–uses–object (multiple aspect).”</td>
</tr>
<tr>
<td>Overview</td>
<td>“The overview abstraction level addresses both multiple layers and multiple aspects.”</td>
</tr>
</tbody>
</table>

Table 4 Grouping of views - Level of abstraction

Displaying a variety of information on multiple layers, such as the subsidiaries (organizational units), the according business applications and their conformity to the architectural standard of the enterprise, the example given represents an overview or coherent type of view.

The formerly explained classification of the views is depicted in the following figure.

---

As highlighted by a red frame, each view is assigned to exactly one group consisting of the two attributes explained before: Level of abstraction and theme.

Overall, there are seven groups of views created as of now. Based on the criteria explained before, these groups are:

<table>
<thead>
<tr>
<th>Theme</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group name</td>
<td>Process High</td>
</tr>
<tr>
<td></td>
<td>Process Low</td>
</tr>
<tr>
<td>Level of abstraction</td>
<td>Overview/Coherence</td>
</tr>
</tbody>
</table>

Table 5 Grouping of views - Theme: Process

<table>
<thead>
<tr>
<th>Theme</th>
<th>Application/Technical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group name</td>
<td>Application High</td>
</tr>
<tr>
<td></td>
<td>Application Low</td>
</tr>
<tr>
<td>Level of abstraction</td>
<td>Overview/Coherence</td>
</tr>
</tbody>
</table>

Table 6 Grouping of views - Theme: Application/Technical

<table>
<thead>
<tr>
<th>Theme</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group name</td>
<td>Architecture High</td>
</tr>
<tr>
<td></td>
<td>Architecture Low</td>
</tr>
<tr>
<td>Level of abstraction</td>
<td>Overview/Coherence</td>
</tr>
</tbody>
</table>

Table 7 Grouping of views - Theme: Architecture

<table>
<thead>
<tr>
<th>Theme</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group name</td>
<td>Organization</td>
</tr>
<tr>
<td>Level of abstraction</td>
<td>Overview/Coherence/Detail</td>
</tr>
</tbody>
</table>

Table 8 Grouping of views - Theme: Organization

The before introduced example view (see Figure 7 "Standard conformity layer", extracted from TUM Pattern Catalog) has to be assigned to the group “Application High”, because it deals with the conformity of business applications, displays multiple layers such as subsidiaries and their business applications as well as their conformity to the architectural standard of the company. Furthermore, it addresses the concerns of the high level management. All other views will be assigned to their respective groups accordingly.

Such a view, assigned to the group of “Application High”, could be of good use for both the Chief Architect and the Chief Operating Officer, with regard to the formerly introduced
business scenario. Obviously, the context of “Cornerstone Care” is slightly different. For instance, the chief architect, trying to reduce the amount of self-developed IT systems, can use this type of view to get an overview of the IT systems used in the company. Therefore, the view should display these IT systems, instead of business applications. Red and green colored rectangles no longer represent the conformity to architectural standards, but illustrate if a system was self-developed or not. In that way, the chief architect is able to extract the needed information very quickly out of the view.

4.3 Concluding remarks

During this project the view catalog serves as a base for the research this project team is conducting. For obvious reasons of size, it cannot contain every single instance from each source dealing with views in Enterprise Architecture Management. Nevertheless, a variety of view types is included, providing a profound overview of differences and similarities in purpose, content and design of views.

Moreover, the classification is a helpful way of simplifying the catalog, maintaining the same level of comprehensiveness without losing valuable information. The chosen procedure surely represents just one possibility out of many to group views, but is based on thorough research. Furthermore, views can be added to the catalog much easier using groups and attributes provided by the classification.

In conclusion both the catalog and the classification of views serve as “stepping stones” for the ultimate goal of creating a domain specific language. The attributes identified in the catalog and used to group views might be employed in the DSL as well. Therefore, they will help during the final steps of this project.


5 Entity Catalog/EAM Entities

By making use of different entities, a view is able to address a concern of a stakeholder. Indeed, entities contain the necessary information that is used in a view. While each stakeholder’s concern may differ from the other one, entities included in a view may also vary from one another view. Therefore, here arises the need to have a well-defined and comprehensive EAM entity catalog that can supply views with high-level entities to address most of the stakeholder’s concerns.

Thereby, aside the view catalog, in this research project it is also built an EAM entity catalog that can support high-level EAM modelling. The main goal of this catalog is to offer a common and standard set of entities used across the enterprise, which can serve as a reference catalog to build different EAM meta-models. This catalog can be further expanded with other entities from within the enterprise to be able to fulfill any of the business requirements.

To go deeper in the “entity” concept presented in the meta-model used in the definition of terms’ chapter, the concepts that surround an entity are presented in the figure below. Moreover, these concepts will be used across the EAM entity catalog.

![Figure 9 Meta-model of concepts of EAM entity catalog](image)

As it can be seen from the figure above, the main concept here is the “entity”. An entity has several attributes and is a part on one of the four layers defined in the entity catalog. Meanwhile, quality entities can be turned into regular entities and inherit the properties of an entity. Finally, each entity can be used in one or more views.

Nowadays, many organizations exist that have adopted EAM as a practice to successfully harmonize activities between business and IT landscape. Mainly, the modelling has been done in different abstraction layers where the need for it arises. The approach followed in this EAM entity catalog is the one of identifying several entities in a very abstract layer and from there, different specializations (drill-downs) can be derived to fulfill the modelling needs
of the organization. Moreover, in this way the complexity of the model is reduced and the meta-model provided could be also easily understood even from business users.

To build this EAM entity catalog, first a brainstorming process has taken part to identify strategic entities that take part in EAM modelling. Furthermore, several related and scientific literatures are evaluated to gather these entities. As conclusion to the literature review, it is important to be mentioned that the research work on this EAM entity catalog is mainly oriented towards TOGAF. However, this catalog does not offer simply a reproduction of TOGAF, but rather a match between different EAM entities across several literatures (TOGAF Content Metamodel\textsuperscript{24}; Enterprise Architecture at Work: Modelling, Communication and Analysis\textsuperscript{25}; The ADOit EAM Meta-model\textsuperscript{26}; Strategic IT Management - A Toolkit for Enterprise\textsuperscript{27}; EAM Pattern Catalog by Buckl, et al.\textsuperscript{28}) and several EAM meta-models in use from industry.

Furthermore, after conducting several interviews with different EAM experts, it was also found that TOGAF was mainly in the core of the EAM practice in their organizations. As mentioned above, such an EAM practice offers high abstraction level of modelling using high-level entities. On the other hand, when building EAM from scratch in an enterprise, there were also suggestions to first identify entities into very granular levels and build specific EAM models; and from there to create the basis for building EAM in the enterprise.

Indeed, this EAM entity catalog is composed of two main parts: entity part and their attributes’ part. As shown in the tables below, entities are divided according to their respective architectural layers. Furthermore, respective attributes, description; commonly used relationships; commonly related entities to such entities; the source where these entities are found; and also the naming differences of the same entities across evaluated literatures are defined for each entity in this catalog.

5.1 Layers

After a thorough analysis in team and evaluation of different literatures, the entities lay on three main architectural layers and are further supported by a ‘Strategic layer’. Through categorization of the entities in each of these architectural layers, it is then easier to answer different stakeholders’ questions (for instance: what technology component supports a business goal? or what applications are related to a specific business process? etc.).

\textsuperscript{24} The Open Group, TOGAF, “TOGAF Content Metamodel.”

\textsuperscript{25} Lankhorst, Enterprise Architecture at Work.

\textsuperscript{26} “The ADOitMethod Manual.”

\textsuperscript{27} Hanschke, Strategic IT Management.

\textsuperscript{28} Buckl et al., “Enterprise Architecture Management Pattern Catalog.”
### Strategic Layer

In this layer, a set of strategic entities is provided, which in the context of EAM, (TOGAF, 2011) suggests that they “… are intended to capture the surrounding context of formal architecture models, including general architecture principles, strategic context that forms input for architecture modeling, and requirements generated from the architecture.”

If one would see this layer from the perspective of TOGAF, the strategic layer would be equivalent to the TOGAF’s Architecture Principles, Vision and Requirements layer.

In this supportive layer, there are several strategic entities such as requirement, gap, constraint, principles and work packages. These entities were also more or less the same with ones used by interviewed EAM experts in their organizations, however with some small differences on them such as target, strategy etc.

Table 9 below shows the structure of the EAM entity catalog with a respective entity from the strategic layer. The full EAM entity catalog can be found in the Appendix part of this paper.

<table>
<thead>
<tr>
<th>EAM Entity Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Layers: Strategic Concepts</td>
</tr>
<tr>
<td>Entity Attributes: ID, Name, Description, Source, Version, Dependency, Hierarchy, Data actuality, Owner</td>
</tr>
<tr>
<td>Description: A quantitative statement of business need that must be met by a particular architecture or work package (TOGAF)</td>
</tr>
<tr>
<td>Entity Common Relationship: requires</td>
</tr>
<tr>
<td>Commons Related Entities: Business Service, Business Function, Business Capability</td>
</tr>
<tr>
<td>Source: EAM TOGAF METAMODEL, METAMODEL ADicot, Enterprise Architecture at Work, Marc Lankhorst</td>
</tr>
<tr>
<td>Named in Other sources: Business Requirement</td>
</tr>
</tbody>
</table>

### Business Layer

As mentioned above, the EAM is combination of activities from the business and IT landscapes. Thus, this layer focuses particularly on the business side of the organization by identifying several entities that include information relevant to the business context.

Such a layer was similarly found in TOGAF, ADOit, Enterprise at Work and other reviewed literatures. Moreover, this architectural layer was also found in the respective organizations of the EAM experts.

<table>
<thead>
<tr>
<th>Business Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business: Business Actor</td>
</tr>
<tr>
<td>ID, Name, Description, Source, Version, Dependency, Hierarchy, Data actuality, Owner</td>
</tr>
<tr>
<td>A person, organization, or system that has a role that initiates or interacts with activities; for example, a sales representative who travels to visit customers.</td>
</tr>
<tr>
<td>EAM TOGAF METAMODEL, METAMODEL ADicot, Enterprise Architecture at Work, Marc Lankhorst</td>
</tr>
<tr>
<td>Business Actor</td>
</tr>
</tbody>
</table>

Table 10 Business Layer

Table 10 shows an example of a business layer entity from the EAM entity catalog. In this architectural layer can be also found entities like business process, function, capability, role, product, business objects and organization unit. Such entities are typical for this architectural layer as they were commonly found across the evaluated literature. Moreover, other business related entities could be easily integrated to this catalog to align it to different organizations.

---

29 The Open Group, TOGAF, “TOGAF Content Metamodel.”

30 Lankhorst, Enterprise Architecture at Work.
Application Layer

Application layer is another important part of the EAM. This layer is the equivalent of TOGAF's Information System Architecture, which indeed includes the application and the data architecture of TOGAF. Furthermore, all of the interviewed EAM experts also identified the importance of this layer when building EAM in their enterprises.

In this EAM entity catalog, this layer consists of several entities like application service, function, component, interface and information objects. Table 11 shows a sample application layer entity from the EAM entity catalog.

![Table 11 Application Layer](image)

While the business layer focused more on the business landscape entities, here the focus is placed on the IT landscape entities. When mentioning IT landscape, IT infrastructure aspects are not considered part of this layer, but rather the IT applications. These IT applications support, control and provide IT capabilities to support the business architecture in maximizing the business value.

Technology Layer

Lastly, the technology layer includes entities that are specifically related to the IT infrastructure. Similar to the previous layers, this layer was also found in the literature of TOGAF, ADOit, “Enterprise Architecture at Work” and other reviewed literatures for this research project.

Entities like physical infrastructure component (device and network), system software, infrastructure service and interface define this layer. Table 12 shows a sample entity for this layer used in the EAM entity catalog.

![Table 12 Technology Layer](image)

While in some literature, physical components of the IT infrastructure were seen as separated entities, in this catalog, they are all grouped under the physical infrastructure component. This grouping of physical components into a single entity allows high abstraction level and possibility for drill-downs into infrastructure specific components.

By interacting with each other, entities in this layer can serve as a basis for implementing IT applications. In more detailed modeling levels, other entities may derive from the actual ones, which can be then also part of this technology layer.

---

31 Lankhorst, Enterprise Architecture at Work.
5.2 EAM Quality Entity

An added feature of this catalog is the quality entity. Aside from being hybrid concepts in-between entities and attributes, their solely purpose is that depending on the modeling needs of the organization, they could turn into a new entity to support in-depth modelling.

As mentioned above, the EAM entity catalog created in this research project is TOGAF-oriented with several adaptions from other sources also. Hence, to better understand the role of the quality entity in the EAM meta-model, the definition of TOGAF is used as a basis for its understanding. TOGAF defines the EAM Meta-model as: “The core meta-model provides a minimum set of architectural content to support traceability across artifacts. Additional meta-model concepts to support more specific or more in-depth modeling are contained within a group of extensions that logically cluster extension catalogs, matrices, and diagrams, allowing focus in areas of specific interest and focus.”

Similar to the concept of “extensions” used in TOGAF definition that help to support specific modelling needs of the enterprise, these extensions are equivalent to the concept of “quality entities” used in the EAM entity catalog. Thus, quality entities like measurements, contract, driver, objective, goal, project and service are identified as quality entities for this catalog. Additional concepts can be added in this part too to fulfill the necessary business requirements.

<table>
<thead>
<tr>
<th>EAM Quality Entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Name</td>
</tr>
<tr>
<td>Measurements</td>
</tr>
</tbody>
</table>

Table 13 Quality Entity

To visualize the usage of a quality entity that turns into a regular entity, let’s refer to the above mentioned business scenario. Initially, the concern of the stakeholder was to get rid of the non-value adding activities from the business processes. Hence, the meta-model for this scenario would follow as in the figure below.

---

32 The Open Group, TOGAF, “TOGAF Content Metamodel.”
In the figure above, the non-value adding activities (KPI) are added as a new entity aside other entities (ex. business process, function, application service, application component and organization unit) in the meta-model.

However, the reason why the above mentioned quality entities are not initially listed in the same way like the other entities in their respective architectural layers is that they do not have the same abstraction level with the actual entities defined in the EAM entity catalog. Thus, their main purpose is to offer drill-down possibilities to the meta-models.

### 5.3 EAM Entity Attributes

In the same line with entities, the same logic is followed with entity attributes. Here, a common set of attributes is identified across the literature and this set is appropriate to be assigned to each of the entities. Besides the description of each attribute, also the according sources are attached to this catalog. A full list of common and shared attributes can be found at the Appendix part.

Table 14 shows the list of common attributes that could fit to each of the entities defined in this EAM entity catalog.

<table>
<thead>
<tr>
<th>Attributes Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EAM Entity Name</strong></td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Source</td>
</tr>
<tr>
<td>Comment</td>
</tr>
<tr>
<td>Filter for Organizational Unit</td>
</tr>
<tr>
<td>Links</td>
</tr>
<tr>
<td>Version</td>
</tr>
<tr>
<td>Dependency</td>
</tr>
<tr>
<td>Hierarchy</td>
</tr>
<tr>
<td>Hierarchy direction</td>
</tr>
<tr>
<td>Data actuality</td>
</tr>
<tr>
<td>Owner</td>
</tr>
</tbody>
</table>

Table 14 Shared Attributes for all entities
In case of further drill-downs into entities, then additional specific attributes can be attached to this list of common attributes.

<table>
<thead>
<tr>
<th>Strategic Concepts</th>
<th>Requirement</th>
<th>Statement of requirement</th>
<th>Statement of what the requirement is, including a definition of whether the requirement shall be met, should be met, or</th>
<th>TOSAI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rationale</td>
<td>Statement of why the requirement exists.</td>
<td>TOSAI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Date of Creation</td>
<td>Day of creating the requirement</td>
<td>Proposal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Acceptance criteria</td>
<td>Statement of what tests will be carried out to ensure that</td>
<td>TOSAI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Constraint</td>
<td>This metamodel entity has only basic attributes.</td>
<td>TOSAI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Principle</td>
<td>Category</td>
<td>The following categories of principle apply: Guiding Principle, Business Principle, Data Principle, Application</td>
<td>TOSAI</td>
</tr>
</tbody>
</table>

Table 15 Specific Attributes for each of entities

5.4 Concluding Remarks

This EAM entity catalog offers a broad range and commonly well-founded EAM entities and their attributes. By working with such commonly used sources in EAM (ex. TOGAF, ADOit etc.), entities in this catalog are easily adaptable with other specific entities that might arise from different business activities.

On the other hand, architectural layers that contain each of the entities are commonly found in several standard EAM practices and literatures and were also validated from the EAM experts’ interviews.

Hence, having these well-founded entities, this EAM entity catalog could serve as a reference for different enterprises to build their EAM metamodels. In cases when the enterprises might have specific modelling needs, it is possible to drill-down each of the given entities into the necessary granular level to be able to fulfill any business requirements.
6 DSL Code Conception and Development

The actual conception of a DSL is the interface between the implementation in a programming language and the research that was conducted before. This chapter emphasizes the train of thoughts from the research to the adaption of known concepts of design and programming. It shows in a structured way which ideas were used to create a foundation for the code implementation.

6.1 Basic conception and derivation

Based on the research and concepts which were created in this paper, the next step is the actual conception of a DSL within the programming language of Groovy. Groovy itself is an object-oriented programming language with strong similarities to Java, so that known concepts like classes and generalization can be adapted and used. The key feature of a DSL is the easy usage without deep knowledge of programming languages.\(^\text{33}\)

![Figure 11 Example Draft of a view\(^\text{34}\)](image)

In this chapter, a “process support map” (derived from EAM Pattern Catalog) covering multiple Business Applications serves as an example starting point for the conception of the DSL. Sample entities, their relations and the actual viewpoint, which explains the meaning of every seen object in the view, is visualized in Figure 11. This example illustrates three different types of entities which are related to each other as well as the group of the view and

---

\(^\text{33}\) Apache Foundation, “Apache Camel: Groovy DSL.”

\(^\text{34}\) Buckl et al., “Enterprise Architecture Management Pattern Catalog,” 105.
the respective stakeholder. In this case, the view is named “Process Support Map” and the corresponding group is “Application High”, meaning that the view shows the relations of applications on a high abstract level. Furthermore, the stakeholder is the chief architect, who has a concern regarding these applications. The bottom part (Visualization Rules) identifies three entities (Business Process, Business Application and Organizational Unit) and shows their relations to each other: Each Business Application supports a business process in a specific organizational unit. For example, within the Process “Customer Relations” the application “Customer Relationship Platform (200)” is used in the organizational units “Headquarter” and “Subsidiary Munich”, whereas the third organizational unit “Subsidiary Paris” uses another business application in this process. Since the view, the viewpoint as well as the entities are instanced in this picture, we can derive the need for classes here. Figure 12 shows an overview of all the classes that are needed for the later implementation. The relations are based on the research in the previous chapters. All classes are explained in detail after this short introduction of the big picture.

Figure 12 Class Overview

The class “View” always has one or more stakeholders and vice versa. Furthermore, a view can have multiple entities and these entities can be used in different views. The view itself is composed of the classes “Group” and “Visualization Rule”. The second class “Visualization Rule” is composed of one or more instances of the class “Visualization Attribute”.

Figure 13 Class Diagram View

Beginning with the view and based on the view catalogue, a view always includes every other class (see Figure 13). It is the frame in which the actual environment of an enterprise can be
modeled. The view includes attributes like the name, description and the source of the view. The last attribute is the version of the view, which can be used to get a better understanding if changes are applied during the modeling of the view. Since the views in this paper are categorized and related to certain groups, the group itself is also an attribute of the view, as well as the viewpoint, which is further named as the set of visualization rules for understanding purposes. Every view is related to only one group and one visualization rule, stated by the one to n relationship. A view can host any number of entities, and every entity can be included in many views. Therefore, we use a one to n relationship between the view class and the entity class. Since there is no other relation between the view and its included objects than the actual existence of these, this paper defines that the view contains the entity. As shown in Figure 13 the class “Stakeholder” is only related to the class view. This class includes the name of the stakeholder as well as the concern the stakeholder has regarding that view. As already stated, a stakeholder can be assigned to multiple views and vice versa.

![Figure 14 Class Diagram Group and Visualization Rule](image)

The class “Group” and its actual name is directly derived from the artifact catalogue and is defined by the combination of the theme and the level of abstraction, which are explained in the chapter of the catalogue. Both theme and level of abstraction are simple attributes in this context. A group can apply to any number of views, but a view can only be part of one group.

As illustrated in Figure 14, the visualization rule (viewpoint) of a view can be seen as a legend, derived from the actual objects (entities, relationships) and their attributes. The visualization rule gives every object a meaning, so that there is no further need of interpretation. The class itself should therefore include the type of the view, regarding the visualization and the shapes and colors that are used. Since the amount and combinations of shapes and colors can be very large the attribute is outsourced into an own class. For example, an entity for a business process can be assigned with a special shape like a blue colored rectangle, if the company uses this combination also in different contexts. To fit the complete picture, the class “Visualization Rule” contains also a name and a textual description.

---

35 see: „4.2 View - Classification”, page 16
The last used class in this conception is the class “Entity”, which has basically four different attributes: The name, a description, a source for the entity and the relationship. The first three attributes work like the attributes in the view class mentioned before. The last attribute, the “RelationshipList” is an ArrayList which contains the relationships between the instanced entities with each other. This means, that an entity can have several relationships with other entities. As seen in Figure 11, an example for that relationship is the connection between the business process entities, which are supported by business applications. The business applications are also related to the organizational unit, meaning that this entity has two different relationships with two other kinds of entities.

In conclusion, these class diagrams of every needed element in the DSL are the foundation for the following DSL conception and the later implementation in groovy. In addition, it is important to state that the used methods will be explained in the respective chapter of the DSL code implementation as well.

6.2 DSL code implementation

The implementation of the DSL code is done by using the Groovy programming language, which can be used by its own or in a Java environment. Groovy programs can be compiled and run standalone or as part of a larger application. They can be invoked from the command line as Groovy scripts or run interactively through the Groovy shell or the Groovy console. The Groovy language is a perfect fit for the use of Domain Specific Languages (DSL), since it is very dynamic and easy to use. The following chapter describes the implementation of the DSL, so that it can be interpreted by the Groovy programming language. This implementation is based on the previous conception of the different classes and their relations with each other. The full implementation of the DSL can be found in Appendix C - Groovy Implementation.36

Class declaration

The first step is to conceptualize the classes of the concepts, which are used to build the DSL in the Groovy programming language. The structure of each class is very similar for each class and usually consists of the function name followed by the unique identifier and the defining attributes.

```groovy
createGroup (groupId) {
```
withName 'Name'
withTheme 'Theme X'
withLevelOfAbstraction 'LevelOfAbstraction X'
withView ([view1, view2])
}

The function `createGroup` is used to bracket a group of views as a form of categorization within the DSL. The group is defined by the following attributes:

- **GroupID**: Unique technical identifier for the view object. Input can be a one string of characters.
- **Name**: Organizational identifier for the group object. Input can be a string of characters.
- **Theme**: Categorizes a group of views into a specific topic e.g. organizational, technical [...].
- **LevelOfAbstraction**: Can be used to define the position of the group in the schema of a meta model.
- **View**: View objects which are supposed to be grouped. The input for this attribute can be multiple view objects.

`createView (viewId) {
    withName 'Name'
    withDescription 'Description text'
    withEntities ([entity1, entity2])
    withGroup group
    withStakeholders ([[‘Stakeholder x name’, ‘Concern x’], [‘Stakeholder y name’, ‘Concern y’]])
    withVersion ‘Version x’
    withSource ‘View source’
    withVisualizationRule vizationRule
}

The function `createView` can be used to invoke a view object, which is a representation of a view within the DSL. The view object is defined by the attributes as corresponding to the view catalogue:

- **ViewID**: Unique technical identifier for the view object. Input can be a one word string of characters.
- **Name**: Organizational identifier for the view object. Input can be a string of characters.
- **Description**: Information about the view and its purpose. Input can be a string of characters
- **Entities**: Corresponding entities, which are used by the view. Input can be entity objects.
- **Group**: When the view is part of a group, it can be referenced in the group attribute. Input can be a group object.
Stakeholders: Individuals or organizational units who are actors in the view or have an interest. Input is one or more strings of characters as an ArrayList.

Version: Describes the current state of the development process of the view.

Source: The literature or document where the source is mentioned. If there is no source, the attribute can be left empty. Input is a string of characters.

VisualizationRule: The VisualizationRule defines the viewpoint of the view. It contains the description of the graphical elements used to draw the viewpoint of the corresponding view. Input is one or more VisualizationRule object.

createVisualizationRule (visualizationRuleId) {
    withName 'Name'
    withDescription 'Description'
    withView view
    withVisualizationType 'Rule type'
    withVisualizationAttributes ([[entity1, 'color1', 'shape1']], [entity2, 'color2', 'shape2'])
}

The createVisualizationRule function is used to create a VisualizationRule object, which can be used to draw the viewpoint of a view. It is defined by the following attributes:

- VisualizationRuleID: Unique technical identifier for the VisualizationRule object. Input can be a one word string of characters.
- Name: Organizational identifier for the VisualizationRule object. Input can be a string of characters.
- Description: Information about the VisualizationRule and its purpose. Input can be a string of characters.
- View: Corresponding views, in which the VisualizationRule is used. Input can be a view object.
- VisualizationType: The VisualizationType contains the form of the view. The Input can be a string of characters.
- VisualizationAttributes: The VisualizationAttribute contains the elements of the viewpoint. One entry connects the entity object with a corresponding color and shape. When every entity of the viewpoint is defined, the VisualizationRule can be leveraged to draw the viewpoint with the help of the DSL.

createEntity (entityId) {
    withName 'Name'
    withAttribute ([[attribute 1 name', 'attribute 1 value'], [attribute 2 name', 'attribute 2 value']])
    withView view
    withRelationship ([[relationship name1', entity1], [relationship name2', entity2]])
}

The function createEntity is used to invoke an entity object, which is the representation of the entity used by a view. The entity object is defined by the attributes:
• **Name**: Organizational identifier for the entity object. Input can be a string of characters.

• **Attribute**: Describing factors of the entity itself e.g. name or version as strings.

• **View**: Corresponding views, in which the entity is used. Input can be a view object.

• **Relationship**: Representation of the link to another entity, which can be used to draw a metamodel of the specified entities. Input can be a Relationship object.

```groovy
def createRelationship(entityFrom, 'relationship name', entityTo)
```

The `createRelationship` function creates a link between two entities and describe in what relations they stand to one another. `entityFrom` describes the source entity whereas `entityTo` describes the target entity. The relationship name describes what kind of relationship the two entities share e.g. governs, contains, is part of etc.

**Groovy code**

To create the actual groovy implementation, the DSL code has to be translated into the groovy object. The following code represents this implementation.

```groovy
def propertyMissing(id) {
    this.getBinding()[id] = id
}
```

The `propertyMissing` function is used to get all objects that are implemented by their ID. This function can be used to get all the objects by their id.

```groovy
def createEntity(id, closure) {
    def temp = new Entity(id)
    this.getBinding()[id] = temp
    closure.delegate = temp
    closure()
}
```

To create actual entities, the `createEntity` function is used. This function creates an entity with an ID, before the closure is delegated to the entity. The closure should be a method from the entity class.

```groovy
def createRelationship(fromEntity, relationshipName, toEntity) {
    fromEntity.withRelationship([[relationshipName, toEntity]])
}
```

The User can create the relationship between two entities by calling the function above with three parameters: The entity which is the start the relationship, the name of the relationship and the entity which is the end of the relationship.
def createView(id, closure) {
    def temp = new View(id)
    this.getBinding()[id] = temp
    closure.delegate = temp
    closure()
}
def createGroup(id, closure) {
    def temp = new Group()
    this.getBinding()[id] = temp
    closure.delegate = temp
    closure()
}
def createVisualizationRule(id, closure) {
    def temp = new VisualizationRule()
    this.getBinding()[id] = temp
    closure.delegate = temp
    closure()
}

All of the functions above work in a very similar way, meaning that one explanation fits to all of these functions. The definition of these functions is strongly related to the function createEntity, meaning that a view, group or visualization rule are created with an “ID” and its methods.

6.3 Proof of Concept

The proof of concept is based on view V-17 from the TU München view catalog. The view uses the entities business process, organizational unit and business application, and leverages every class from the class declaration chapter in a practical application. The purpose of this proof of concept is to show on an actual example how the groovy implementation can be used to create a textual view with the entities mentioned above. The content of this view can be read in Chapter 6.1.

Imagine the project team (introduced in the Business Scenario Chapter 3) wants to address some of the concerns they have regarding the elimination of non-value adding activities, reducing the amount of different IT-systems and the reduction of self-developed programs. The view V17 visualizes, which business applications support which business processes at which organizational units\(^\text{37}\), and thus is a good starting point to get a thorough overview over the IT-Landscape and interrelations between the IT-systems, allowing the identification of target systems, which can be replaced or optimized to support the automatization of

\(^{37}\text{Buckl et al., “Enterprise Architecture Management Pattern Catalog,” 105.}\)
company workflows by reducing systems or programs which are rarely used, inefficient or obsolete.

```plaintext
createEntity (businessProcess) {
    withName 'CR'
    withAttribute ([['description','Customer Relations'], ['hierarchy', 'no']])
}
```

First, the `createEntity` function is used to create a business process entity with the ID `businessProcess`. Furthermore, some attributes are added to that entity: The name “CR”, the description (“Customer Relations”) as well as the “hierarchy”. Of course, the id `businessProcess` can be used to be called in other DSL snippet as well.

```plaintext
cREATEView (viewV17) {
    withName 'Process Support Map'
    withEntities ([businessProcess])
    withDescription 'This V-Pattern visualizes, which business applications support which business processes at which organizational units.'
    withVersion 'V 1.0'
    withSource 'TUM'
    withStakeholders ([['Chief architect', 'Eliminate NVA from business processes']])
}
```

The “viewV17” view can be created with the `createView` function. It can have a name, description, version and source as textual attributes. The `businessProcess` entity which has been created before is saved in the corresponding ArrayList which includes all entities so far. One exemplary stakeholder with the name “Chief architect” and his concern “Eliminate NVA from business processes” can be created at the same time with the view.

```plaintext
cREATEEntity (organizationalUnit) {
    withName 'HQ'
    withAttribute ([['description','Headquarter'], ['hierarchy', 'yes']])
    withView viewV17
}
```

This `organizationalUnit` entity is created to demonstrate the possibility of adding entities to an existing view when the entity is created. In this case, the entity `organizationalUnit` is added to the view “viewV17”.

```plaintext
cREATERelationship businessProcess, 'used at', organizationalUnit
```
With at least two entities created, the function `createRelationship` can be used to create a relationship between the existing entities. In this example the relationship `used at` is created between the entity `businessProcess` and `organizationalUnit`.

```plaintext
createEntity (businessApplication) {
    withName 'CRP'
    withAttribute ([['description', 'Customer Relationship Platform'], ['hierarchy', 'no']])
    withRelationship ([['supported', businessProcess]])
}
```

As seen above, another entity `businessApplication` is created with simultaneously adding a relationship between said entity and the existing entity `businessProcess`. The relationship is in this case named `supported`.

```plaintext
createGroup (groupAPH) {
    withName 'Application High'
    withTheme 'Application'
    withLevelOfAbstraction 'High'
    withView ([viewV17, viewV16])
}
```

As a next step, the group `groupAPH` is created with its name ("Application High"), theme ("Application") and level of abstraction ("High"). Furthermore, another view ("viewV16") is included in that same group.

```plaintext
createVisualizationRule (rule1) {
    withName 'Rule 1'
    withDescription 'Rule no.1'
    withVisualizationType 'Rule type 1'
    withVisualizationAttributes ([[businessApplication, 'white', 'round-rectangle'], [organizationalUnit, 'blue', 'round-rectangle'], [businessProcess, 'blue', 'step-shape']])
    withView viewV17
}
```

Finally, a visualization rule can be created with some text properties like name, description, visualization type. The visualization attributes can be created with an `ArrayList` of many combinations of entity, color and shape. View "viewV17" is the view where this exact rule is
applied in. The full implementation of the DSL can be found in Appendix C - Groovy Implementation. This Groovy Implementation includes

a) Sample creation implementation of
   a. Entities (business process, organizationalUnit, businessApplication)
   b. Views (viewV16, viewV17, viewV18)
   c. Relationship (“used at”)
   d. Group (groupAPH)
   e. (Dummy) Visualization Rules (rule1, rule2)

b) Definitions of methods (createEntity, createRelationship, createView, createGroup, createVisualizationRule)

c) Definitions of classes (View, Entity, EntityAttribute, Relationship, Stakeholder, VisualizationRule, VisualizationAttribute, Group)
7 Conclusion

In this project, a DSL was developed which enables stakeholders to create views (including entities) and address their company related concerns. Furthermore, it can be used in order to describe EAM metamodels. To achieve this goal and create a useful practical DSL, the project team analyzed EAM specific literature to identify commonly used entities and views and summarized the results in an entity and view catalog. The view catalog does not comprise each view from the EAM sector, but provides a profound overview of differences and similarities in terms of purpose, content, and design of commonly used views. Within the catalog, a classification was developed, which reduced the degree of complexity and provided an understandable structure without losing valuable information. The Entity Catalog offers a wide range of continuously used entities and attributes which were taken from sources such as TOGAF and the EAM tool ADoIT. To proof the practical usefulness, a business scenario called “Cornerstone Care” was defined, which describes a typical business case in the EAM sector. Furthermore, it was described how to address typical issues supported by the Catalogs of Entities and Views. Thereby, the catalogs served as a basis for creating the DSL covering typical attributes of views and entities. Furthermore, an EAM metamodel visualizes the relationships between the different EAM concepts. One benefit of using a EAM metamodel is, that the relationships of entities are shown comprehensibly for the business side. Moreover, the result can also be used as a basis for further research about common EAM entities and views.

A concept was created based on the catalogs, from which concrete code snippets were shown in order to create views and entities. By using the business scenario, the practical usefulness of the DSL was verified to some extent. The main insights are, that the DSL is characterized by its simple structure. So the DSL can be introduced easily in a company. In this way the work process can be optimized and misunderstandings due to communication issues can prevented in advance. From these insights, it can be concluded, that this DSL might serve as an interface between the business and technical side. Moreover, the language can be used as a standard in companies.

For further development, the DSL could be proofed in terms of reliability and usability. Therefore, an information modelling platform, like Socio Cortex, could be used in order to connect its semantic platform as backend of the DSL (interpreter). Thereby, it could be proofed whether the usage of the DSL can contribute to better communication between business and IT side and improve collaboration.
References


## Appendix

Appendix A  
A-1 Team organization  
A-2 Project plan  

Appendix B  
B-1 View Catalog  
B-2 EAM Entity Catalog  
B-3 EAM Quality Entities  
B-4 EAM Entities Attributes Catalog  
B-5 Quality Entity Attributes  

Appendix C  
Groovy Implementation
Appendix A  Appended information

A-1  Team organization

In this project, two teams exchanged ideas, progress and worked together through continuous predefined meetings.

<table>
<thead>
<tr>
<th>Team Entities</th>
<th>Team Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person 1 (Secretary)</td>
<td>Person 6 (Inter-team Coordinator)</td>
</tr>
<tr>
<td>Person 2 (Spokesperson)</td>
<td>Person 7 (Chief Editor)</td>
</tr>
<tr>
<td>Person 3 (Mediator)</td>
<td>Person 8</td>
</tr>
<tr>
<td>Person 4 (Project Manager)</td>
<td>Person 9</td>
</tr>
<tr>
<td>Person 5 (Inter-team coordinator)</td>
<td></td>
</tr>
</tbody>
</table>

Team members with respective roles

The task definition and responsibility for the roles “Inter-Team Coordinator” and “Chief Editor” will be stated as follows:

**Inter-Team Coordinator**

- is responsible for communication between groups (via contact person of other group, e.g. weekly meetings)
- communicates process status of other team
- needs to be up to date in terms of statuses of current team work
- possible problems should be identified early and communicated within the group and/or professor

**Chief Editor**

- is responsible for layout, formatting and editing of written report
- manages time schedule and structure for the written report
- collects texts (subchapters, content for appendix etc.) from responsible authors and makes content available for all participants
- aligns and compiles text fragments to one coherent report
- administrates collaboration and citation tools/databases and hosts introductory courses
- contact person for all further report-related issues
Organizational commitment and Report Review

Besides previous mentioned points, further commitments were taken. These included the preparation of bi-weekly status meetings, reminders of inter-team agreements, pursuit and demand of work results, identification and handling of unplanned activities/tasks, etc.

Furthermore, a fundamental quality control/assurance for the report in the end was carried out. Due to wide range of “writing habits” and quality standards, individual arrangements for rewriting of paragraphs or other adjustments of textual content were unavoidable.

In addition to the temporary support of some members, the inter-team coordinator of Team Views and the Chief Editor took responsibility for these parts during the entire project and its completion.

A-2 Project plan

The planning of this project is guided by three presentations serving as milestones for the project progress:

- *The Kick-off presentation*: serving as an effective start of the project, in which the team discussed questions related to: organization, planning and tasks.
- *The Intermediate presentation*: is the halfway of the project, all team members present their deliveries (catalogs + metamodels) and discuss the final steps that should be taken.
- *The final presentation*: is a presentation of the final results and demonstration of the DSL developed.

The following chart provides a visualized overview.
### Project Plan

<table>
<thead>
<tr>
<th>Activity</th>
<th>Dates</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction to Topic:</strong></td>
<td>17 - 09 - 2015</td>
</tr>
<tr>
<td>Gathering Source materials</td>
<td>17 - 09 - 2015</td>
</tr>
<tr>
<td>Preparing for the Kick-off</td>
<td>22 - 10 - 2015</td>
</tr>
<tr>
<td><strong>Kick-off Presentation:</strong></td>
<td>22 - 10 - 2015</td>
</tr>
<tr>
<td>Research and deliveries</td>
<td>22 - 10 - 2015</td>
</tr>
<tr>
<td>Surveys (Interviews)</td>
<td>22 - 10 - 2015</td>
</tr>
<tr>
<td>First version of Catalogs (Entities &amp; Views)</td>
<td>22 - 10 - 2015</td>
</tr>
<tr>
<td><strong>Intermediate presentation:</strong></td>
<td>12 - 01 - 2016</td>
</tr>
<tr>
<td>Research on groovy (DSL)</td>
<td>12 - 01 - 2016</td>
</tr>
<tr>
<td>Creation of DSL - Entities</td>
<td>12 - 01 - 2016</td>
</tr>
<tr>
<td>Creation of DSL - Views</td>
<td>12 - 01 - 2016</td>
</tr>
<tr>
<td>Integration of modules</td>
<td>12 - 01 - 2016</td>
</tr>
<tr>
<td><strong>Final Presentation:</strong></td>
<td>04 - 04 - 2016</td>
</tr>
</tbody>
</table>

**Appendix**
Appendix B  Tables
## B-1 View Catalog

<table>
<thead>
<tr>
<th>Name of the view(point)</th>
<th>Version</th>
<th>Purpose</th>
<th>Group</th>
<th>Description</th>
<th>Visualization</th>
<th>Source</th>
<th>Main entities</th>
<th># entities</th>
<th>Stakeholder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of the view(point)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name of the view(point)</td>
<td>Version</td>
<td>According to Lankhorst: Designing, Deciding, Informing</td>
<td>Process High/Low Application High/Low Organization High/Low Application High/Low</td>
<td>Description of content/purpose, e.g. summary of TUM purpose/content</td>
<td>Based on legends in TUM Pattern Catalog: E.g. Map symbols and Visualization rules</td>
<td>E.g. &quot;Viewpoint V-5&quot; from TUM</td>
<td>E.g. Business application, Software system</td>
<td></td>
<td>Number of entities: What stakeholders (or their concerns) are addressed by the view?</td>
</tr>
</tbody>
</table>

| Standard Conformity Layer | 1.0 | Deciding, Informing | Application High | This pattern visualizes conformity (interpreted in a dichotomous way as yes or no) aspects to company standards. | Rectangles symbolize Organizational Units which frame Business Applications | Viewpoint V-5 from TUM | Organizational Unit, Business application | 28 | (High) management level, e.g. CEO, CFO etc. OR Business analyst |

<p>| Business Process and Business Function Relationship | 1.0 | Informing, Deciding | Process High | This pattern gives an overview of the business events, the business processes they trigger, and the business functions, e.g. organizational units, responsible for the processes. | Circular areas symbolize business functions, events and processes. (Directed) lines show coherence relationships like 'reacts to' or 'proceses' between two business processes. | Viewpoint V-12 from TUM | Business function, Business event, Business process | 9 | Business analyst |
| Service-based Business Process Support Map | 1.0 | Designing, Informing | Process Low | This pattern visualizes, how business processes are supported by services provided by business applications. | Rectangles symbolize Business Applications which have directed arrows to Application Services which get exposed by them. These services are represented by circular areas and have themselves directed arrows (symbolize &quot;supports&quot; relationship) to Business Processes. | Viewpoint V-18 from TUM | Business process, Application service, Business application | 14 | Business analyst, business process manager |
| Technologies by Architectural Standard | 1.0 | Informing | Architecture Low | Pattern consists of a table containing the technologies used in an architectural solution. | A table consisting of two columns (Name, Used Technologies) provides a list of used technologies in an architectural solution. | Viewpoint V-23 from TUM | Name, Used technologies | 2 | IT architect |
| Time Interval Map visualizing Lifecycles of Applications | 1.0 | Deciding, Informing | Application Low | Pattern visualizes the lifecycles of business applications, including the respective versions. | A grid representing a time span (x-axis) and Business Applications with different application versions (y-axis). The intersection contains of multiple application status (planned, in development, in production, in retirement) | Viewpoint V-26 from TUM | Business application (Version), Application (Version) status planned, in development, in production and in retirement | 35 | Business analyst |
| Clustering [business applications] by Standard | 1.0 | Deciding, Informing | Architecture High | Pattern visualizes homogeneity aspects, concerning architectural solutions. | Architectural solutions (incl. version), illustrated by rectangles, framing matching business applications (also illustrated by rectangles). | Viewpoint V-6 from TUM | Architectural solution, Business application (with ID) | 33 | Chief architect |</p>
<table>
<thead>
<tr>
<th>Knowledge Needs</th>
<th>Viewpoint</th>
<th>Knowledge set, Degree of knowledge coverage in %</th>
<th>Business analyst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deciding Pattern visualizes the existing knowledge for different programming languages.</td>
<td>V-8 from TUM</td>
<td>2</td>
<td>CIO (?), IT-related decision maker, but rather not chief architect</td>
</tr>
<tr>
<td>Process Support Map</td>
<td>Viewpoint</td>
<td>Business process, Business application, Organizational unit</td>
<td>Business analyst</td>
</tr>
<tr>
<td>Designing, Deciding, Informing Pattern visualizes, which business applications support which business processes at which organizational units.</td>
<td>V-17 from TUM</td>
<td>47</td>
<td>CEO, High level management</td>
</tr>
<tr>
<td>Cluster Map for hosting relationship</td>
<td>Viewpoint</td>
<td>Organizational Unit, Business application</td>
<td>CEO, High level management</td>
</tr>
<tr>
<td>Deciding, Informing Application High V-Pattern visualizes organizational units hosting business applications</td>
<td>V-24 from TUM</td>
<td>32</td>
<td>CEO, High level management</td>
</tr>
<tr>
<td>Cluster Map for using relationship</td>
<td>Viewpoint</td>
<td>Organizational Unit, Business application</td>
<td>CEO, High level management</td>
</tr>
<tr>
<td>Deciding, Informing Application High V-Pattern visualizes organizational units using business applications</td>
<td>V-25 from TUM</td>
<td>32</td>
<td>CEO, High level management</td>
</tr>
<tr>
<td>Application Lifecycle Project Layer</td>
<td>Viewpoint</td>
<td>Business application (Version), Project, Application (Version) status planned, in development, in production and in retirement</td>
<td>Business analyst</td>
</tr>
<tr>
<td>Deciding, Informing Application Low V-Pattern visualizes project proposals in addition to the lifecycles of the business applications</td>
<td>V-27 from TUM</td>
<td>37</td>
<td>Business analyst</td>
</tr>
<tr>
<td>Process Support Map visualizing horizontal integration</td>
<td>1.0</td>
<td>Designing, Deciding, Informing</td>
<td>Application High</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>-----</td>
<td>---------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Process Support Map visualizing vertical integration</td>
<td>1.0</td>
<td>Designing, Deciding, Informing</td>
<td>Application High</td>
</tr>
<tr>
<td>Process Support Map visualizing vertical and horizontal integration</td>
<td>1.0</td>
<td>Designing, Deciding, Informing</td>
<td>Application High</td>
</tr>
<tr>
<td>Process Support Map visualizing changes in relations to their time horizon</td>
<td>1</td>
<td>Deciding, Informing</td>
<td>Application High</td>
</tr>
<tr>
<td>Realization of requirements</td>
<td>1</td>
<td>Informing, Deciding (?)</td>
<td>Process High</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---</td>
<td>-------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Stakeholder viewpoint</td>
<td>1</td>
<td>Informing, Deciding</td>
<td>Organization High</td>
</tr>
<tr>
<td>Goal realization viewpoint</td>
<td>1</td>
<td>Informing</td>
<td>Process Low</td>
</tr>
<tr>
<td>Goal contribution viewpoint</td>
<td>1</td>
<td>Informing, Deciding (?)</td>
<td>Process High</td>
</tr>
<tr>
<td>Principles viewpoint</td>
<td>1</td>
<td>Designing</td>
<td>Process High</td>
</tr>
<tr>
<td>---------------------</td>
<td>---</td>
<td>-----------</td>
<td>--------------</td>
</tr>
<tr>
<td>Motivation viewpoint</td>
<td>1</td>
<td>Information</td>
<td>Process High</td>
</tr>
<tr>
<td>Project viewpoint</td>
<td>1</td>
<td>Information, Deciding</td>
<td>Architecture High</td>
</tr>
<tr>
<td>Migration viewpoint</td>
<td>1</td>
<td>Information, Deciding</td>
<td>Architecture High</td>
</tr>
<tr>
<td><strong>Implementation and Migration viewpoint</strong></td>
<td>1</td>
<td>Information</td>
<td>Architecture High</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>Introductory Viewpoint</strong></td>
<td>1</td>
<td>Information</td>
<td>Architecture High</td>
</tr>
<tr>
<td><strong>Organization Viewpoint</strong></td>
<td>1</td>
<td>Information, Deciding</td>
<td>Organization High/Low</td>
</tr>
<tr>
<td>Viewpoint</td>
<td>Action</td>
<td>Process Level</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Actor cooperation Viewpoint</td>
<td>1</td>
<td>Information, Deciding</td>
<td>Process Low</td>
</tr>
<tr>
<td>Business Process Viewpoint</td>
<td>1</td>
<td>Deciding, Informing, Designing</td>
<td>Process High</td>
</tr>
</tbody>
</table>
## Appendix B-2  EAM Entity Catalog

<table>
<thead>
<tr>
<th>ID</th>
<th>Entity Layers</th>
<th>Entity Name</th>
<th>Entity Attributes</th>
<th>Description</th>
<th>Commonly Relationship Name</th>
<th>Commonly Related Entities</th>
<th>Source</th>
<th>Named in Other sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Strategic Concepts</td>
<td>Requirement</td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>A quantitative statement of business need that must be met by a particular architecture or work package. <em>(TOGAF)</em></td>
<td>requires</td>
<td>Business Service, Business Function, Business Capability</td>
<td>EAM TOGAF CONTENT METAMODEL, METAMODEL ADOIT, Marc Lankhorst</td>
<td>Business Requirement</td>
</tr>
<tr>
<td>2</td>
<td>Constrains</td>
<td></td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>An external factor that prevents an organization from pursuing particular approaches to meet its goals. <em>(TOGAF)</em></td>
<td>impact</td>
<td>Business Service, Requirements</td>
<td>EAM TOGAF CONTENT METAMODEL, Marc Lankhorst</td>
<td>Constrains</td>
</tr>
<tr>
<td>3</td>
<td>Principles</td>
<td></td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>A qualitative statement of intent that should be met by the architecture. Has at least a supporting rationale and a measure of importance. <em>(TOGAF)</em></td>
<td>governs</td>
<td>Business Service</td>
<td>EAM TOGAF CONTENT METAMODEL, Marc Lankhorst, METAMODEL ADOIT, Strategic IT Management</td>
<td>Business Principles <em>(TOGAF)</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>A statement of difference between two states. Used in the context of gap analysis, where the difference between the Baseline and Target Architecture is identified. (TOGAF)</td>
<td>-</td>
<td>-</td>
<td>EAM TOGAF CONTENT METAMODEL</td>
<td>Gap</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>4</td>
<td>Gap</td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>A set of actions identified to achieve one or more objectives for the business. A work package can be a part of a project, a complete project, or a program. (TOGAF)</td>
<td>achieves</td>
<td>Requirements, Project, Business Actor, Business Process, Business Capability</td>
<td>EAM TOGAF CONTENT METAMODEL, METAMODEL</td>
<td>Work Package</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Work Package</td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>A person, organization, or system that has a role that initiates or interacts with activities; for example, a sales representative who travels to visit customers. Actors may be internal or external to an organization. (TOGAF)</td>
<td>supplies, consumes, belongs to, participates in, perform, perform task in, interacts with</td>
<td>Business Function, Organizational Unit, Business Process, Business Role, Business Service, Information Object</td>
<td>EAM TOGAF CONTENT METAMODEL, METAMODEL</td>
<td>Business Actor</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Business</td>
<td>Business Actor</td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>A unit of internal behaviour or collection of causally-related units of internal behaviour intended to produce a defined set of products and services. The name of a business process should preferably be or contain a verb in the simple present tense, e.g. ‘Receive request’. (Mark Lankhorst)</td>
<td>involves, accessed by, decomposes, participates, guided by, precedes, follows, orchestrates, produces</td>
<td>Business Actor, Business Service, Business Function, Product, Organization Unit</td>
<td>EAM TOGAF CONTENT METAMODEL, METAMODEL</td>
<td>Business Process</td>
</tr>
<tr>
<td>7</td>
<td>Business Process</td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>A statement of difference between two states. Used in the context of gap analysis, where the difference between the Baseline and Target Architecture is identified. (TOGAF)</td>
<td>-</td>
<td>-</td>
<td>EAM TOGAF CONTENT METAMODEL</td>
<td>Gap</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>Category</td>
<td>Description</td>
<td>Business Function</td>
<td>Business Service</td>
<td>Business Capability</td>
<td>Product</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Business Function</td>
<td>A unit of internal behaviour that groups behaviour according to (for example) required skills, knowledge, resources, etc. The name of a business function should preferably be or contain a verb ending with 'ing', e.g. 'Claims processing'. (Mark Lankhorst)</td>
<td>Supports business capabilities through an explicitly defined interface and is explicitly governed by an organization. (TOGAF)</td>
<td>provided to, governed by, measured by, provides, consumes, realized by, implemented on, traced again, owned by, supports, realized by, meets</td>
<td>Supports business capability that is delivered by the completion of one or more work packages. (TOGAF)</td>
<td>Output generated by the business. The business product of the execution of a process. (TOGAF)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Business Role</td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>A named specific behaviour of a business actor participating in a particular context. The actor perform the role. (Mark Lankhorst)</td>
<td>accesses, performed by Business Actor, Business Function</td>
<td>EAM TOGAF CONTENT METAMODEL, METAMODEL ADOIT, TUM Pattern Catalog, Marc Lankhorst</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Business Objects</td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>A business object represents a real-world entity – abstract or concrete – which encapsulates some part of the business activity of an enterprise (customers, for example, products or orders). Business objects can be associated with one another by relationships. An “order” can be decomposed into sub-objects such as “order header” and “order content”. (Inge Hanschke)</td>
<td>accessed by, provides, supplied by, consumed by Business Service, Business Function, Business Actor</td>
<td>Strategic IT management, Marc Lankhorst, METAMODEL ADOIT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Organization Unit</td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>A self-contained unit of resources with goals, objectives, and measures. Organization units may include external parties and business partner organizations. (TOGAF)</td>
<td>owns, contains, participates, governs, produces, motivated by, operates in Business Actor, Driver, Business Function, Product, Business Service</td>
<td>EAM TOGAF CONTENT METAMODEL, METAMODEL ADOIT, TUM Pattern Catalog, Marc Lankhorst, Strategic IT management</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Business Role

Business Objects

Business Unit (Inge Hanschke)
<p>| 15 | Application | Application Service | ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner | An externally visible unit of functionality, provided by one or more components, exposed through well-defined interfaces, and meaningful to the environment. (Marc Lankhorst) | used by, provides, implements, realizes, realized on | Business Actor, Business Service, Application Component | EAM TOGAF CONTENT METAMODEL, METAMODEL ADOiT, TUM Pattern Catalog, Marc Lankhorst | Information System Service(TOGAF) |
| 16 | Application Function | Application Function | ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner | A coherent unit of internal behaviour of an application component. (Marc Lankhorst) | supports, realized by, bounded by, owned by, accessed by, consumes | Application Service, Application Component, Infrastructure Service, Information Objects | Marc Lankhorst, METAMODEL ADOiT | Application Function |
| 17 | Application Component | Application Component | ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner | A modular, deployable, and replaceable part of a system that encapsulates its contents and exposes its functionality through a set of interfaces. (Marc Lankhorst) | accesses, implemented on, implements, operates on, extended by | Information Objects, Physical Infrastructure, System Software, Application Service, Application Interface, Infrastructure Interface, Business Functions | EAM TOGAF CONTENT METAMODEL, METAMODEL ADOiT, TUM Pattern Catalog, Marc Lankhorst, Strategic IT management | Business Application (TUM Pattern Catalog) |
| 18 | Interface | Interface | ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner | Declares how a component can connect with its environment ex. how is accessed an application. (Marc Lankhorst) | accessed by, relates to | Business Role, Application Component, Application Service | METAMODEL ADOiT, Marc Lankhorst, Strategic IT Management, TUM Pattern Catalog | Interface |</p>
<table>
<thead>
<tr>
<th>19</th>
<th>Information Objects</th>
<th>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</th>
<th>Information objects are artefacts specific to applications. They have relations with business objects which represent the overarching business artefacts agreed in conceptual design. (Inge Hanschke)</th>
<th>accessed by, provides, supplied by, updated by, updated through, relates to</th>
<th>Application Service, Business Service, Application Component</th>
<th>EAM TOGAF CONTENT METAMODEL, METAMODEL ADOiT, TUM Pattern Catalog, Marc Lankhorst, Strategic IT management</th>
<th>Data Entity (TOGAF), Data Objects (Marc Lankhorst)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Technology</td>
<td>Physical Infrastructure Component (Device and Network)</td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>Consisting of all the physical components of the technology layer such as device and network. (Self-Created)</td>
<td>realizes, extends</td>
<td>System Software</td>
<td>EAM TOGAF CONTENT METAMODEL, METAMODEL ADOiT, TUM Pattern Catalog, Marc Lankhorst, Strategic IT management</td>
</tr>
<tr>
<td>21</td>
<td>System Software</td>
<td>A software environment for specific types of application components and data objects that are deployed on it in the form of artifacts. (Marc Lankhorst)</td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>accessed by, extended by, provides platform for</td>
<td>Application Component, Application Service, Infrastructure Service, Physical Infrastructure Component</td>
<td>Marc Lankhorst, METAMODEL ADOiT, TUM Pattern Catalog</td>
<td>Logical Technology Component (TOGAF)</td>
</tr>
<tr>
<td>22</td>
<td>Infrastructure Service</td>
<td>Externally visible unit of functionality, provided by one or more nodes, exposed through well-defined interfaces, and meaningful to the environment. (Marc Lankhorst)</td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>realized by, supplied by</td>
<td>System Software</td>
<td>EAM TOGAF CONTENT METAMODEL, METAMODEL ADOiT, TUM Pattern Catalog, Marc Lankhorst</td>
<td>Platform Service (TOGAF), Infrastructure System (Inge Hanschke)</td>
</tr>
<tr>
<td>No.</td>
<td>Interface</td>
<td>ID, Name, Description, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>A point of access where the infrastructural services offered by a node can be accessed by other nodes or by application components. (Marc Lankhorst)</td>
<td>Physical Infrastructure Component, Application Component, Infrastructure Service</td>
<td>METAMODEL ADOiT, Marc Lankhorst, Strategic IT Management, TUM Pattern Catalog</td>
<td>Infrastructure Interface (Marc Lankhorst)</td>
<td></td>
</tr>
</tbody>
</table>
## B-3 EAM Quality Entities

<table>
<thead>
<tr>
<th>ID</th>
<th>Entity Name</th>
<th>Entity Attributes</th>
<th>Description</th>
<th>Commonly Related Name</th>
<th>Commonly Related Entities</th>
<th>Source</th>
<th>Named in other sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Contract</td>
<td>ID, Name, Description, Category, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>An agreement between a service consumer and a service provider that establishes functional and non-functional parameters for interaction. (TOGAF)</td>
<td>signed by, approved by, rejected by, govern, measure, meet</td>
<td>Business Service, Service Quality, Business Actor</td>
<td>EAM TOGAF CONTENT METAMODEL, METAMODEL ADOiT</td>
<td>Contract</td>
</tr>
<tr>
<td>2</td>
<td>Driver</td>
<td>ID, Name, Description, Category, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>An external or internal condition that motivates the organization to define its goals. (TOGAF)</td>
<td>creates, address, motivates</td>
<td>Goal, Organization Unit</td>
<td>EAM TOGAF CONTENT METAMODEL, METAMODEL ADOiT</td>
<td>Driver</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ID, Name, Description, Category, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>A time-bounded milestone for an organization used to demonstrate progress towards a goal. (TOGAF)</td>
<td>realizes, achieved by(KPIs), aligned to, traced against</td>
<td>Goal, Measures</td>
<td>EAM TOGAF CONTENT METAMODEL, METAMODEL ADOiT</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>Objectives</td>
<td></td>
<td>A high-level statement of intent or direction for an organization. Typically used to measure success of an organization. (TOGAF)</td>
<td>related to, realized by, addresses</td>
<td>Driver, Objectives</td>
<td>EAM TOGAF CONTENT METAMODEL, METAMODEL ADOiT, Strategic IT Management</td>
<td>Goal</td>
</tr>
<tr>
<td>4</td>
<td>Goal</td>
<td></td>
<td>Actual transformations of the EA are performed by projects. A project is a group of activities and tasks that change the EA, fulfilling an agreed set of objectives and success measures.</td>
<td>implements, fulfill, traced against, produces</td>
<td>Business Service, Product, Objectives, Measures</td>
<td>METAMODEL ADOiT, Strategic IT Management, TUM Pattern Catalog</td>
<td>Project</td>
</tr>
</tbody>
</table>
There are at least two states for a project: ‘approved’ and ‘proposed’.

<table>
<thead>
<tr>
<th>ID</th>
<th>Measurements</th>
<th>Description</th>
<th>Category</th>
<th>Source</th>
<th>Comment</th>
<th>Filter for Organizational Unit</th>
<th>Links</th>
<th>Version</th>
<th>Dependency</th>
<th>Hierarchy</th>
<th>Data actuality</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Measurements</td>
<td>ID, Name, Description, Category, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>An indicator or factor that can be tracked, usually on an ongoing basis, to determine success or alignment with objectives and goals. (TOGAF)</td>
<td>trace against, set, aligned to, measures</td>
<td>Objectives, Business Service</td>
<td>EAM TOGAF CONTENT METAMODEL, METAMODEL ADOiT, Strategic IT Management</td>
<td>KPI (Inge Hanschke)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Service Quality</td>
<td>ID, Name, Description, Category, Source, Comment, Filter for Organizational Unit, Links, Version, Dependency, Hierarchy, Hierarchy direction, Data actuality, Owner</td>
<td>A preset configuration of non-functional attributes that may be assigned to a service or service contract. (TOGAF)</td>
<td>applies to, assigned to, controls</td>
<td>Contract, Business Service</td>
<td>EAM TOGAF CONTENT METAMODEL</td>
<td>Service Quality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# EAM Entities Attributes Catalog

<table>
<thead>
<tr>
<th>EAM Entity Name</th>
<th>Name of Attribute</th>
<th>Description of Attribute</th>
<th>Source of Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Unique identifier for the architecture entity</td>
<td>TOGAF, ADOit</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Brief name of the architecture entity</td>
<td>TOGAF, ADOit, Hanschke</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Textual description of the architecture entity.</td>
<td>TOGAF, ADOit, Hanschke</td>
<td></td>
</tr>
<tr>
<td>Source</td>
<td>Location from where the information was collected.</td>
<td>TOGAF</td>
<td></td>
</tr>
<tr>
<td>Comment</td>
<td>Other remarks</td>
<td>ADOit</td>
<td></td>
</tr>
<tr>
<td>Filter for Organizational Unit</td>
<td>Defines relevance of business units</td>
<td>ADOit</td>
<td></td>
</tr>
<tr>
<td>Links</td>
<td>Links to external data</td>
<td>ADOit</td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>version of entity ex. 1.0, 1.1, 1.2</td>
<td>Bente, ADOit</td>
<td></td>
</tr>
<tr>
<td>Dependency</td>
<td>requires or replaces</td>
<td>Bente</td>
<td></td>
</tr>
<tr>
<td>Hierarchy</td>
<td>yes or no</td>
<td>Bente</td>
<td></td>
</tr>
<tr>
<td>Hierarchy direction</td>
<td>up or down</td>
<td>Proposal</td>
<td></td>
</tr>
<tr>
<td>Data actuality</td>
<td>Date of update</td>
<td>ADOit</td>
<td></td>
</tr>
<tr>
<td>Owner</td>
<td>Owner of the architecture entity.</td>
<td>TOGAF</td>
<td></td>
</tr>
<tr>
<td>Requirement</td>
<td>Statement of requirement</td>
<td>TOGAF</td>
<td></td>
</tr>
<tr>
<td>Rationale</td>
<td>Statement of why the requirement exists.</td>
<td>TOGAF</td>
<td></td>
</tr>
<tr>
<td>Date of Creation</td>
<td>Day of creating the requirement</td>
<td>Proposal</td>
<td></td>
</tr>
<tr>
<td>Acceptance criteria</td>
<td>Statement of what tests will be carried out to ensure that the requirement will be met.</td>
<td>TOGAF</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>No additional attributes</td>
<td>This metamodel entity has only basic attributes.</td>
<td>TOGAF</td>
</tr>
<tr>
<td>Principle</td>
<td>Category</td>
<td>The following categories of principle apply: Guiding Principle, Business Principle, Data Principle, Application Principle, Integration Principle, Technology Principle.</td>
<td>TOGAF</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Priority</td>
<td>Priority of this principle relative to other principles.</td>
<td>TOGAF</td>
<td></td>
</tr>
<tr>
<td>Statement of principle</td>
<td>Statement of what the principle is.</td>
<td>TOGAF</td>
<td></td>
</tr>
<tr>
<td>Rationale</td>
<td>Statement of why the principle is required and the outcome to be reached.</td>
<td>TOGAF</td>
<td></td>
</tr>
<tr>
<td>Implication</td>
<td>Statement of what the principle means in practical terms.</td>
<td>TOGAF</td>
<td></td>
</tr>
<tr>
<td>Gap</td>
<td>No additional attributes</td>
<td>TOGAF</td>
<td></td>
</tr>
<tr>
<td>Work Package</td>
<td>Category</td>
<td>The following categories of work package apply: Work Package, Work Stream, Project, Program, Portfolio.</td>
<td>TOGAF</td>
</tr>
<tr>
<td>Date of Creation</td>
<td>Day of creating the requirement</td>
<td>Proposal</td>
<td></td>
</tr>
<tr>
<td>Capability delivered</td>
<td>Describes the contribution this work package makes to capability delivery.</td>
<td>TOGAF</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Business Architecture</th>
<th>Business Actor</th>
<th>Actor goal</th>
<th>Objectives that this actor has, in general terms.</th>
<th>TOGAF</th>
</tr>
</thead>
<tbody>
<tr>
<td>#FTE</td>
<td>Number of FTE that operate as this actor.</td>
<td>TOGAF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actor tasks</td>
<td>Tasks that this actor performs, in general terms.</td>
<td>TOGAF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process criticality</td>
<td>Criticality of this process to business operations.</td>
<td>TOGAF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manual or automated</td>
<td>Whether this process is supported by IT or is a manual process.</td>
<td>TOGAF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process volumetrics</td>
<td>Data on frequency of process execution.</td>
<td>TOGAF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Service</td>
<td>Starting Date</td>
<td>Starting Day of the Process</td>
<td>Proposal</td>
<td></td>
</tr>
<tr>
<td>End Date</td>
<td>Ending Day of the Process</td>
<td>Proposal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Capability</td>
<td>Business value</td>
<td>Describes how this capability provides value to the enterprise.</td>
<td>TOGAF</td>
<td></td>
</tr>
<tr>
<td>Increments</td>
<td>Lists possible maturity/quality levels for the capability.</td>
<td>TOGAF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Function</td>
<td>No additional attributes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product</td>
<td>Production Date</td>
<td>Date of product's production</td>
<td>Proposal</td>
<td></td>
</tr>
<tr>
<td>Expiry Date</td>
<td>If yes, the date of expiration. If no, can be left empty.</td>
<td>Proposal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Role</td>
<td>Estimated number of FTEs that operate in this Role</td>
<td>This metamodel entity has only basic attributes.</td>
<td>TOGAF</td>
<td></td>
</tr>
<tr>
<td>Organization Unit</td>
<td>Headcount</td>
<td>Number of FTEs working within the organization.</td>
<td>TOGAF</td>
<td></td>
</tr>
<tr>
<td>Application Architecture</td>
<td>Business Object</td>
<td>No additional attributes</td>
<td>Description</td>
<td>Hanschke</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------</td>
<td>--------------------------</td>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td>Application Interfaces</td>
<td>Planning status</td>
<td>The planning status of the interface; as for applications this can be “as-is”, “planned” and “to-be”</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Utilisation period</td>
<td>Period over which the interface is used in productive operation (“from-to” specification)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Assignment to applications</td>
<td>Which applications are linked by this interface?</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Assignment to Informations object</td>
<td>Which information objects are captured by this interface?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information Objects</td>
<td>Assignment to applications</td>
<td>The applications to which the information object is assigned, including how the applications use the object.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Assignment to interfaces</td>
<td>The interfaces to which the information object is assigned, and the flow direction (information flow).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applications Component</td>
<td>Planning status</td>
<td>The planning status of the application: this can be “as-is”, “planned” or “to-be”.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Assignment to interfaces</td>
<td>An indication of which interfaces the application has.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Size of application</td>
<td>The size of application.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SLA requirements and references to SLAs</td>
<td>SLA requirements such as availability, performance or downtime, details on SLA agreements such as contract numbers and periods of validity.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Line of Code</td>
<td>Number of lines of code.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Assignment to Informations object</td>
<td>Which information objects are used, and the nature of this usage, e.g. “transactional data”?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Responsible Actor</td>
<td>Can be assigned to individuals, to groups or to entire organisations. Individual responsibilities are business and technical responsibilities (such as the application manager, business manager, or business contact person). Groups and organisations can be entities such as the support organisation or business organisation. It is also possible to include specific details here, e.g. you can indicate who is responsible for assigning authorisations.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Differentiate between third-party or individual application component, and also the nature of the use: “OLTP”, “OLAP”, or the nature of the operation interface: “web-based”, “rich client” and “host client”.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security level</td>
<td>For example: low, medium and high.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Vendor</strong></td>
<td><strong>Number of Users</strong></td>
<td><strong>Ex SAP</strong></td>
<td><strong>Hanschke</strong></td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------------------</td>
<td>-----------------------------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td><strong>External Applications</strong></td>
<td><strong>Number of users of this application component</strong></td>
<td><strong>ADOit</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>In development since</strong></td>
<td><strong>Other related external applications to this application component</strong></td>
<td><strong>ADOit</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Utilisation period</strong></td>
<td><strong>The period over which the application is used. For applications with “as-is” status, this will be the period of time over which the application is in productive use. For “planned” or “to-be” applications, this is a statement on the planned period of use.</strong></td>
<td><strong>Hanschke</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dialog or Background application</strong></td>
<td><strong>Is it a dialog or background application?</strong></td>
<td><strong>ADOit</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Business criticality</strong></td>
<td><strong>Criticality of this application to business operations.</strong></td>
<td><strong>ADOit</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Top Application</strong></td>
<td><strong>Yes or No</strong></td>
<td><strong>ADOit</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Input Data</strong></td>
<td><strong>What is the input data to the application</strong></td>
<td><strong>Proposal</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Output Data</strong></td>
<td><strong>What is the output data of the application</strong></td>
<td><strong>Proposal</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Application Service</strong></td>
<td><strong>No additional attributes</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Application Function</strong></td>
<td><strong>No additional attributes</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Technology Architecture**

<table>
<thead>
<tr>
<th>Physical Infrastructure Component</th>
<th>Vendor</th>
<th>Name of the vendor of the physical component.</th>
<th><strong>TOGAF</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td></td>
<td>Categorisation according to the TOGAF TRM, which may be extended to meet the needs of an individual organization. See TOGAF TRM.</td>
<td><strong>TOGAF</strong></td>
</tr>
<tr>
<td>System Software</td>
<td>Category</td>
<td>Categorisation according to the TOGAF TRM, which may be extended to meet the needs of an individual organization. See TOGAF TRM.</td>
<td><strong>TOGAF</strong></td>
</tr>
<tr>
<td>Vendor</td>
<td>License Expiry Date</td>
<td>Date of license expiry</td>
<td><strong>Proposal</strong></td>
</tr>
<tr>
<td></td>
<td>Operating System</td>
<td>Running Operating System (Windows, Linux etc.)</td>
<td><strong>Proposal</strong></td>
</tr>
<tr>
<td></td>
<td>Assignment to applications</td>
<td>Applications to which this software is assigned.</td>
<td><strong>Proposal</strong></td>
</tr>
<tr>
<td></td>
<td>In development since</td>
<td>if yes, since when</td>
<td><strong>ADOit</strong></td>
</tr>
<tr>
<td></td>
<td>Language</td>
<td>In what languages is the software?</td>
<td><strong>ADOit</strong></td>
</tr>
<tr>
<td>Infrastructure service</td>
<td>Category</td>
<td>Criticality of this software to business operations.</td>
<td>ADOit</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------</td>
<td>-----------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Top Software</td>
<td>Yes or No</td>
<td>Platform Services are categorized according to the TOGAF TRM, which may be extended to meet the needs of an individual organization. See TOGAF TRM</td>
<td>Proposal</td>
</tr>
<tr>
<td>Infrastructure Interface</td>
<td>Planning status</td>
<td>The planning status of the interface; as for infrastructure this can be “as-is”, “planned” and “to-be”</td>
<td>TOGAF</td>
</tr>
<tr>
<td></td>
<td>Utilisation period</td>
<td>Period over which the interface is used in productive operation (“from-to” specification)</td>
<td>Proposal based on Hanschke (application interface)</td>
</tr>
<tr>
<td>Assignment to physical infrastructure component</td>
<td>Which infrastructures are linked to this interface.</td>
<td>Proposal based on Hanschke (application interface)</td>
<td></td>
</tr>
</tbody>
</table>
### Quality Entity Attributes

<table>
<thead>
<tr>
<th></th>
<th>Characteristics</th>
<th>Description of different behavior, security, availability, scalability etc. characteristics of the contract.</th>
<th><strong>TOGAF</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Contract</strong></td>
<td><strong>Characteristics</strong></td>
<td><strong>Description of different behavior, security, availability, scalability etc. characteristics of the contract.</strong></td>
<td><strong>TOGAF</strong></td>
</tr>
<tr>
<td>Service times</td>
<td>Hours during which the service must be available</td>
<td><strong>Description of different behavior, security, availability, scalability etc. characteristics of the contract.</strong></td>
<td><strong>TOGAF</strong></td>
</tr>
<tr>
<td>Date of start</td>
<td>Starting date of contract</td>
<td><strong>Description of different behavior, security, availability, scalability etc. characteristics of the contract.</strong></td>
<td><strong>Proposal</strong></td>
</tr>
<tr>
<td>Date of end</td>
<td>Ending date of contract</td>
<td><strong>Description of different behavior, security, availability, scalability etc. characteristics of the contract.</strong></td>
<td><strong>Proposal</strong></td>
</tr>
<tr>
<td>Responsible Unit</td>
<td>Organizational Unit responsible for the contract</td>
<td><strong>Description of different behavior, security, availability, scalability etc. characteristics of the contract.</strong></td>
<td><strong>Proposal</strong></td>
</tr>
<tr>
<td><strong>Driver</strong></td>
<td>No additional attributes</td>
<td><strong>Description of different behavior, security, availability, scalability etc. characteristics of the contract.</strong></td>
<td><strong>TOGAF</strong></td>
</tr>
<tr>
<td><strong>Objectives</strong></td>
<td>No additional attributes</td>
<td><strong>Description of different behavior, security, availability, scalability etc. characteristics of the contract.</strong></td>
<td><strong>TOGAF</strong></td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td>No additional attributes</td>
<td><strong>Description of different behavior, security, availability, scalability etc. characteristics of the contract.</strong></td>
<td><strong>TOGAF</strong></td>
</tr>
<tr>
<td><strong>Project</strong></td>
<td>No additional attributes</td>
<td><strong>Description of different behavior, security, availability, scalability etc. characteristics of the contract.</strong></td>
<td><strong>TOGAF</strong></td>
</tr>
<tr>
<td><strong>Measures (KPI)</strong></td>
<td>No additional attributes</td>
<td><strong>Description of different behavior, security, availability, scalability etc. characteristics of the contract.</strong></td>
<td><strong>TOGAF</strong></td>
</tr>
<tr>
<td><strong>Service Quality</strong></td>
<td>No additional attributes</td>
<td><strong>Description of different behavior, security, availability, scalability etc. characteristics of the contract.</strong></td>
<td><strong>TOGAF</strong></td>
</tr>
</tbody>
</table>
Appendix C  Groovy Implementation

note: This Groovy Implementation includes

d) Sample creation implementation of
   a. Entities (business process, organizationalUnit, businessApplication)
   b. Views (viewV16, viewV17, viewV18)
   c. Relationship (“used at”)
   d. Group (groupAPH)
   e. (Dummy) Visualization Rules (rule1, rule2)

e) Definitions of methods (createEntity, createRelationship, createView, createGroup, createVisualizationRule)

f) Definitions of classes (View, Entity, EntityAttribute, Relationship, Stakeholder, VisualizationRule, VisualizationAttribute, Group)

package DSLGroovy

/**
 * @author KieuDaiTran
 */

//demo
createEntity (businessProcess) {
    withName 'CR'
    withAttribute (['description','Customer Relations'], ['hierarchy', 'no'])
}

ccreateView (viewV17) {
    withName 'Process Support Map'
    withEntities ([businessProcess])
    withDescription 'This V-Pattern visualizes, which business applications support which business processes at which organizational units.'
    withVersion 'V 1.0'
    withSource 'TUM'
    withStakeholders (['Chief architect', 'Eliminate NVA from business processes'])
}

createEntity (organizationalUnit) {
    withName 'HQ'
    withAttribute (['description','Headquarter'], ['hierarchy', 'yes'])
    withView viewV17
}

createRelationship businessProcess, 'used at', organizationalUnit

createEntity (businessApplication) {
    withName 'CRP'
    withAttribute (['description','Customer Relationship Platform'], ['hierarchy', 'no'])
    withView viewV17
    withRelationship ([['supported',businessProcess]])
}

ccreateView (viewV16) {
    withName 'View V16'
    withDescription 'View V16 desc'
    withVersion 'V 1.0'
    withSource 'TOGAF'
}
withStakeholders (['CIO', 'concern1'], ['manager', 'concern2'])
}

createGroup (groupAPH) {
    withName 'Application High'
    withTheme 'Application High'
    withLevelOfAbstraction 'High'
    withView ([viewV17, viewV16])
}

createVisualizationRule (rule1) {
    withName 'Rule 1'
    withDescription 'Rule no.1'
    withVisualizationType 'Rule type 1'
    withVisualizationAttributes (['businessApplication', 'white', 'round-rectangle'],
                                ['organizationalUnit', 'blue', 'round-rectangle'],
                                ['businessProcess', 'blue', 'step-shape'])
    withView viewV17
}

createVisualizationRule (rule2) {
    withName 'Rule 2'
    withDescription 'Rule no.2'
    withVisualizationType 'Rule type 2'
}

ccreateView (viewV18) {
    withName 'View V18'
    withDescription 'View V18 desc'
    withVersion 'V 1.1'
    withVisualizationRule rule2
    withGroup groupAPH
}

println businessProcess
println organizationalUnit
println businessApplication
println viewV17
println viewV18
println groupAPH
println rule1
println rule2

def propertyMissing(id) { this.getBinding()[id] = id }

def createEntity(id, closure) {
    def temp = new Entity(id)
    this.getBinding()[id] = temp
    closure.delegate = temp
    closure()
}

def createRelationship(fromEntity, relationshipName, toEntity) {
    fromEntity.withRelationship([[relationshipName, toEntity]])
}

def createView(id, closure) {
    def temp = new View(id)
    this.getBinding()[id] = temp
    closure.delegate = temp
    closure()
}

def createGroup(id, closure) {
    def temp = new Group()
    this.getBinding()[id] = temp
    closure.delegate = temp
    closure()
}
def createVisualizationRule(id, closure) {
    def temp = new VisualizationRule()
    this.getBinding()[:id] = temp
    closure.delegate = temp
    closure()
}

class View {
    //property
    String id
    String name
    String description
    String version
    String source
    VisualizationRule rule
    Group group
    def entities = []
    def stakeholders = []

    //constructor
    View(id) {this.id = id}

    //methods
    def withName(name) {
        this.name = name
    }

    def withDescription(desc) {
        this.description = desc
    }

    def withVersion(version) {
        this.version = version
    }

    def withSource(source) {
        this.source = source
    }

    def withVisualizationRule(rule) {
        this.rule = rule
        rule.view = this
    }

    def withGroup(group) {
        this.group = group
        group.views << this
    }

    def withEntities(entitiesList) {
        for (entity in entitiesList) {
            entity.view = this
            entities << entity
        }
    }

    def withStakeholders(stakeholdersArray) {
        for (stakeholder in stakeholdersArray) {
            this.stakeholders << new Stakeholder(name: stakeholder[0], concern: stakeholder[1])
        }
    }

    String toString() {
        String result = "
View: " + name
        result += "
Description: " + description
        result += "
Version: " + version
        result += "
Source: " + source
        return result
    }
}
result += "\n\tVisualization Rule: " + rule.name
result += "\n\tGroup: " + group.name
result += "\n\tStakeholders:
for (obj in stakeholders) {
    result +="\n\t" + obj.name + " - " + obj.concern
} result += "\n\tEntities:
for (obj in entities) {
    result +="\n\t" + obj.name
}
return result
}

class Entity{
    // property
    String id
    View view
    String name
    def attributes=[]
    def relationships=[]

    // constructor
    Entity(id) { this.id = id }

    // methods
    def withName(name){
        this.name = name
    }

    def withAttribute(attList) {
        for (att in attList) {
            def attribute = new EntityAttribute(name: att[0], value: att[1])
            attributes << attribute
        }
    }

    def withRelationship(relationshipsList) {
        for (relationship in relationshipsList) {
            this.relationships << new Relationship(name: relationship[0], from: this, to: relationship[1])
        }
    }

    def withView(view){
        this.view = view
        view.entities << this
    }

    String toString() {
        String result = "\nEntity: " + name
        result += "\nAttributes:" for (obj in attributes) {
            result +="\n\t" + obj.name + ": " + obj.value
        }
        result += "\nRelationships:" for (obj in relationships) {
            result +="\n\t" + obj.name + ": " + obj.to.name
        }
        if (view) result += "\nView: " + view.name
        return result
    }
}
class EntityAttribute{
    String name
class Relationship{
    String name
    Entity from
    Entity to
}

class Stakeholder{
    String name
    String concern
}

class VisualizationRule{
    String name
    String description
    String visualizationType
    View view
    def visualizationAttributes = []

    //methods
    def withName(name){
        this.name = name
    }

    def withDescription(desc){
        this.description = desc
    }

    def withVisualizationType(type){
        this.visualizationType = type
    }

    def withVisualizationAttributes(attributes){
        for (att in attributes){
            this.visualizationAttributes << new VisualizationAttribute(entity: att[0],
                                                                          color: att[1], shape:att[2])
        }
    }

    def withView(view){
        this.view = view
        view.rule = this
    }

    String toString() {
        String result = "VisualizationRule: " + name
        result += "\n\tDescription: " + description
        result += "\n\tVisualization Type: " + visualizationType
        result += "\n\tView: " + view.name
        result += "\n\tVisualization Attributes:
        for (obj in visualizationAttributes) {
            result +="\n\t\t" + obj.entity.name + ": " + obj.color + " - " + obj.shape
        }
        return result
    }
}

class VisualizationAttribute{
    String shape
    String color
    Entity entity
}

class Group{
    //properties
String name
String theme
String levelOfAbstraction
def views=[]

//methods
def withName(name){
    this.name = name
}
def withTheme(theme){
    this.theme = theme
}
def withLevelOfAbstraction(loa){
    this.levelOfAbstraction = loa
}
def withView(views){
    for (view in views){
        view.group = this
        this.views << view
    }
}

String toString() {
    String result = "\nGroup: " + name
    result += "\nTheme: " + theme
    result += "\nLevel Of Abstraction: " + levelOfAbstraction
    result += "\nViews:
    for (obj in views) {
        result +="\n  " + obj.name
    }
    return result
}