Modeling Techniques for Enterprise Architecture Documentation: Experiences from Practice

Thomas Trojer, Matthias Farwick, and Martin Haeusler

University of Innsbruck, Innsbruck, Austria
`firstname.lastname@uibk.ac.at`

Abstract. Enterprise Architecture Management (EAM) is an IT-management process in which the relationships of business services, applications and the underlying IT-infrastructure is modeled. With dedicated EA models, activities such as architectural consolidation, planning and risk analysis are facilitated. A key factor in modeling and documenting enterprise architectures is the the underlying meta-model that enables to capture the information demand of an organization. Current EA tools provide no or only inflexible mechanisms to create and evolve such organization-specific meta-models. Therefore we present a novel modeling framework that has been established from a consulting and a research project with two data centers. It makes use of both, concepts from multi-level modeling and classical three-level modeling and separates structural and ontological model ingredients. A key aspect of the presented approach is the separation of modeling tasks across different stakeholders and modeling levels and the favoring of practical usability over language feature richness.

1 Introduction

In the context of *Enterprise Architecture Management* (EAM) and *(IT-)systems operation management*, specialized modeling tools are typically used to model the dependencies between the IT-infrastructure, deployed applications and the business functions they support [14]. These models are then used to analyze the current architecture, assess risks and plan changes to it.

In our previous work [7] we showed that keeping such a model in-sync with reality is a major problem in practice. In line with Schweda [16], we argue that a key aspect of an organization's ability to effectively utilize and update the model, is to create an evolvable organization-specific meta-model that matches the stakeholders current information demand. In the context of EAM we call this meta-model the *information model* (i.e. *M1*).

An information model is defined to only capture organization relevant data and specifies the architectural patterns that occur in the organization's business and IT. However, the proper definition of such a model is hard to achieve in practice: Typically multiple types of modeling artifacts need to be maintained and different stakeholders use and adjust them. This poses challenges to both, the underlying modeling framework itself as well as the user interfaces that manage certain stakeholder groups to only model those parts for which they have expertise.

In this paper we briefly present the EA and IT-modeling tool *Txture*[1] that, among other features, is capable of a flexible creation of information models and the maintenance of instances thereof. The main purpose of *Txture* is to support the architecture documentation efforts of stakeholders in an enterprise. It separates the modeling concerns at different modeling layers via dedicated user interfaces and thus links the different stakeholder groups to their individual area of expertise and responsibility. The tool is the result of ongoing consulting and research work in collaboration with two data centers that helped us to identify modeling challenges in practice.

A central part of this paper is to show how we tackled modeling challenges with a combination of a classical three-level modeling approach, a type – instance based modeling approach and flexible extensions for individual model elements. The goal of work is to share our experiences from practice and to engage in discussions with the multi-level modeling research community on our approach and potential alternative methods.

The remainder of this paper is structured as follows: We first provide general background information for the *Txture*-tool. We then continue by presenting IT-architecture modeling requirements and challenges in Section 3. In Section 4, we describe how our modeling framework tackles these. Finally, we discuss related work and end with concluding remarks.

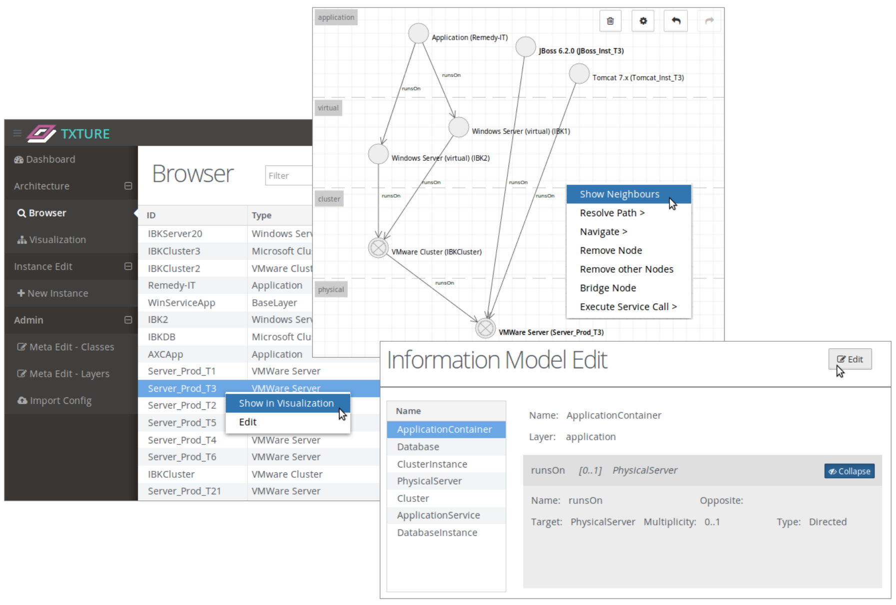
2 Background of the IT-Modeling Tool Txture

In 2011 we started a consulting project with a banking data center and subsequently a research project with the data center of a large semiconductor manufacturer. The overall goal of both projects was to make IT-infrastructure documentation more efficient and effective. Enhanced usability features and stakeholder-orientation of the implemented tool were generally considered important. Besides, requirements of common work activities on top of an IT-documentation, such as flexible visualizations of the architecture for planning and risk analysis, have been elicited.

The key features of the resulting *Txture* modeling tool are:

– Dynamic and flexible visualizations of IT-architectures via graphs.
– Configurable import mechanisms to automatically use architectural data contained in external sources such as in *Configuration Management Databases* (CMDB), *Excel* spreadsheets or relational databases.
– Modeling of the architecture via a form-based web-client to support less technically skilled users.
– Textual architecture modeling via an information-model aware *Eclipse*-based text editor [8].

[1] `http://www.txture.org`



ID	Type
IBKServer20	Windows Serv
IBKCluster3	Microsoft Clu
IBKCluster2	VMware Clust
Remedy-IT	Application
WinServiceApp	BaseLayer
IBK2	Windows Serv
IBKDB	Microsoft Clu
AXCApp	Application
Server_Prod_T1	VMWare Server
Server_Prod_T3	VMWare Server
Server_Prod_T2	
Server_Prod_T5	
Server_Prod_T4	VMWare Server
Server_Prod_T6	VMWare Server
IBKCluster	VMware Cluster
Server_Prod_T21	VMWare Server

Name
ApplicationContainer
Database
ClusterInstance
PhysicalServer
Cluster
ApplicationService
DatabaseInstance

Name: ApplicationContainer

Layer: application

runsOn *[0..1] PhysicalServer*

Name: runsOn Opposite:

Target: PhysicalServer Multiplicity: 0..1 Type: Directed

Fig. 1. The *Txture* environment showing the architecture browser (left screenshot), navigable visualizations (top-most) and the ability to view and change the information model (bottom-most). The tool is fully functional.

– High-performance model queries via optimized persistence of models in a graph database.
– The ability to define and change the information model at runtime.

In order to exemplify the motivation for such a tool, Figure 1 depicts a graph-based architecture visualization in *Txture*. Here, relationships between *application containers*, an *application* and the underlying (clustered) hardware infrastructure are shown. Such a visualization is used in practice e.g., to perform impact and risk analysis of application deployments.

Several other key visualization features can be seen in this figure:

– Architectural elements are assigned to configurable layers, hence the visualization automatically shows an intuitive architectural stack.
– The visualization is navigable via a set of traversal, deletion and grouping operations for depicted documentation nodes (see context menu in Figure 1).
– Nodes are styled based on their type or other attributes, like mission-criticality.

Furthermore, Figure 1 also shows the meta-modeling capabilities for defining information models via a form-based editor.

In the following section we outline the main modeling challenges that led to *Txture*'s underlying modeling framework.

3 Modeling Challenges

Related work in the domains of IT-systems modeling as well as in the multi-level modeling community has mostly focused on modeling software systems (refer to Section 5). Compared to the modeling of software systems, IT-infrastructure documentation has unique modeling requirements and its very own challenges. We can summarize the most important challenges as follows:

CH1: *Separation of stakeholders defining the information model and the ones who are actually responsible for documenting IT-systems:* Two main modeling stakeholder groups were determined in both projects: One group consists of *Expert Architects* and the other group are *Element Responsibles.* Expert architects have an interest in overseeing the entire IT-architecture of a given organization. This group of stakeholders usually works together in order to develop the organization-specific information model that forms the basis for the actual IT-systems documentation. Architects are not necessarily the persons who document the architecture. This is the task of *Element Responsibles*, who maintain dedicated parts of the architecture documentation. Opposed to the architects, they are mostly experts in a very narrow field that revolves around the items and technologies they work with. *Server* responsibles are likely to be experts in very specific types of virtualizations or hardware. *Application* responsibles, on the other hand, often only roughly know the hardware their applications are deployed on. As one can see, distinct user groups perform modeling on different levels. This challenge needs to be tackled by a proper IT-documentation tool.

CH2: *Documentation and information models both need to be evolvable:* In cases where the general architectural structure of an organization shifts, it becomes necessary to adapt the information model (think e.g. introduction of cloud computing), or an architectural pattern was discovered that can not be documented with the current information model. These types of changes should be applicable while the tool is running and without the help of a modeling expert (i.e. no recompilation, complex configuration and adherence to certain modeling patterns is required).

CH3: *A documentation tool needs to expose familiar terminology and enterprise-aligned concepts to stakeholders:* Especially in the more technology-related parts of an architecture documentation, changes occur frequently and require updates to modeling artifacts. A documentation tool needs to enable stakeholders to quickly re-establish a useful documentation that is in-sync with the real world and reflects enterprise-specific terminology.

CH4: *Documentations need to be extensible according to individual stakeholder's documentation requirements:* Different stakeholders typically have different documentation demands. We found out e.g. that components which are central to an IT-architecture are likely to be documented in a more detailed way than others. Therefore a documentation tool needs to flexibly cater for documentation intents of stakeholders that are beyond a defined common model.

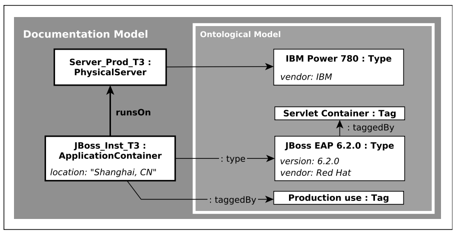


Fig. 2. A simple IT-infrastructure documentation model.

4 Modeling Solutions

In this section we further outline the *Txture* modeling framework, first, by providing an example model on which we base our discussions and second, by presenting our solution approach to tackle the abovementioned challenges.

The *documentation model* (i.e. *M0*) in Figure 2 shows instances of IT-system components that are documented. The specific example describes an application container instance "*JBoss_Inst_T3*" which "*runs on*" a physical server named "*Server_Prod_T3*". As we have described in the previous section, such a documentation model can be used e.g., to perform impact analysis ("What happens if the specific server crashes?") or to do infrastructure planning ("Is the specific server appropriately dimensioned to run such software?"). Additional to modeling IT-component instances and their structural dependencies, a simple notion of *ontology* can be seen on the right side of the figure. Such ontological classifications are modeled as part of the documentation activity (also on *M0*) and allow *Element responsibles* to further describe and categorize their documented instances.

The descriptive concepts which are available are *types* and *tags*. Types are used to provide a (domain-related) classification of instances and may also define additional attributes for them. In our example case, the application container instance is of type *"JBoss EAP 6.2.0"*. Additionally, a set of tags can be assigned to instances and types. They provide a simple means to further classify elements via keywords. In our example the server type is tagged *"Servlet Container"* to indicate its relatedness to *Java servlet* technology. The typing and tagging mechanisms are also used in *Txture* to allow browsing, search and filter functionality across the IT-systems documentation.

Figure 3 provides an extended picture of our example model by including its meta-model hierarchy. On the information model level, the expressiveness of the underlying documentation model is set. At this level the *linguistic structure* that architects agreed upon is modeled. Opposed to this, the *ontological structure*, which reflects domain expert knowledge, is modeled as part of the documentation process. This reflects the separation that is demanded in challenge **CH1**.

The top-level artifact, the *meta-meta model* (i.e. *M2*), defines all concepts that are used in *Txture* and, in line with requirements of our industry partners, are needed to properly describe their IT-infrastructures. It defines the concepts

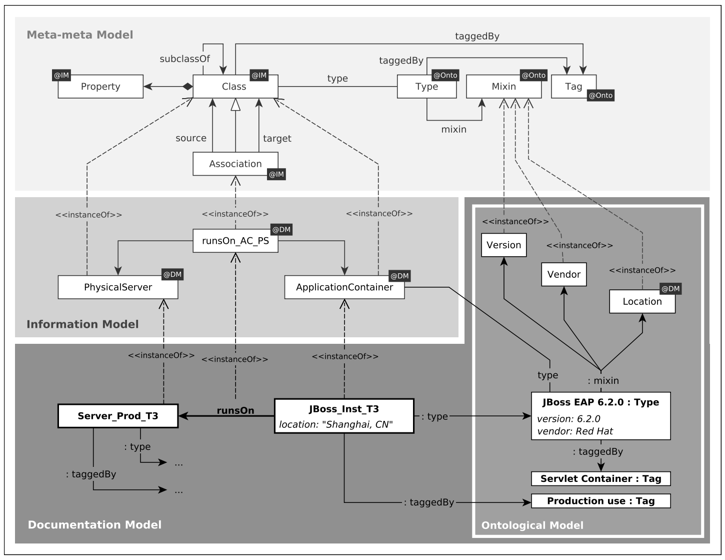


Fig. 3. The *Txture* modeling infrastructure. Annotation boxes (black) reflect where a model element gets instantiated (@IM = Information Model, @Onto = Ontological model and @DM = Documentation Model).

class, *association* (i.e. association classes) and *property* to develop the structure of an organization-specific architecture modeling language (i.e. the linguistic model) and the concepts *type*, *tag* and *mixin* that shape the ontological model.

4.1 Classical Hierarchies to separate Modeling Activities

One of the experiences we gained from modeling workshops with our industry partners is that modeling novices or software developers understand modeling best when using strict and limited hierarchies in which modeling concepts and their instantiations are described. In our case the modeling levels that users have to interact with are manifested by the information model and the documentation model as its instantiation.

Besides understandability of concepts, having a clear cut between modeling levels also supports a permission and concern-oriented separation for managing the IT-documentation and the information model it relies on. This separation is important as different modeling activities are performed by individual stakeholders with potentially diverse domain expertise. In one of our projects, stakeholder roles like employees from operations, database administrators, software developers and also project managers were involved in the documentation process and a few selected stakeholders of these groups together with the IT-architects managed the information model.

In our tool the modeling of each level is separated by different user interface and therefore tackles challenge **CH1**.

4.2 Types to mitigate Invasive Information Model Changes

Another experience we made was that adapting the information model is typically a recurring activity, triggered by frequent change requests from industry partners and driven by adjustments, extensions and simplifications to modeled concepts. It is common to any modeling activity, that changes to models may involve corresponding changes on dependent models, as part of re-establishing conformance in the model hierarchy. To minimize the efforts and consequences of such changes, either well-defined automated model refactoring procedures are required or an information model needs to be realized in a way so that the most-common changes to it only minimally interfere. For our industry partners a manual refactoring after information model changes was out of question. This is why we settled on a modeling pattern similar to the one of *power types* [15] that allows for creating types at the documentation model level and therefore reduces the need to actually adapt the related information model.

Our original modeling approach made heavy use of inheritance on the information model level. For example we applied a deep inheritance structure to model different *application containers* according to their *vendor*, *software version* or required *runtime platform*. This rendered the information model both, large in size (i.e. number of model elements) and prone to frequent changes (e.g. on software version changes).

Using *types* greatly helped to reduce the size of the information model and therefore maintaining comprehensibility and lowering the frequency in which changes to it needed to be applied. Based on using types, our new modeling approach consists of only including generic information model elements like *physical server* or *application container*. The goal was to provide basic, but stable modeling concepts that are *invariant* to an organization. These concepts span the structure of the model, i.e. the permissible nodes and relations between them. This reflects the generic structure that all involved stakeholders can relate to. E.g. no highly-specific vendor-based product terminology is used that would only be understood by a minority of stakeholders. Accordingly, in our ontological model we allow to extend information model elements with the help of *types*. Types are part of the documentation model, but reference elements of the information model. In the example of Figure 2 and 3 *JBoss EAP 6.2.0* extends the meaning of the documented instance *JBoss_Inst_T3* beyond that of being an *application container*. While *application container* can be considered a stable information model concept, *JBoss*-specific server software will likely change from time to time and by our understanding of types can be easily adjusted within the documentation model. This is in line with Atkinson and Kühne [2], who describe the need for changes and newly added types that are possible while the system is running. Our type concept delivers a light-weight way for dynamic additions and proved to be intuitively usable in IT-infrastructure documentation practice.

In addition to types, we use *tags* to further categorize documentation model elements. Tags are comparable to *UML stereotypes*[2] and can be applied to types

[2] cf. *UML 2.4.1* infrastructure specification, `http://www.omg.org/spec/UML/2.4.1/`

and individual instances. In *Txture* both, type and tag elements are modeled by element responsibles and as part of the documentation model.

Our intention with types is to reduce the amount of changes on the information model, tackling challenge **CH2**. Using types together with tags as a means to categorize and describe documented instances contributes a solution to challenge **CH3**.

4.3 Multi-level Instantiation to support Dynamic Extensions

With the introduction of types on the documentation model level, we are able to limit the amount of changes that otherwise are applied to the information model. While this is beneficial, maintaining an information model of only generic concepts bares issues regarding the expressiveness of the documentation: Generic information model concepts leave out detail and shift the specification of properties of documentation elements onto types. Our documentation activities require that types and instances can be managed by the same stakeholders within the documentation model. For proper infrastructure documentation, types not only define properties to be instantiated by their related instances, but need to specify values for certain properties themselves.

Figure 3 shows that the *JBoss*-example type defines values for the properties *version* and *vendor*, whereas our example application container defines a text value reflecting its deployment *location* to be “Shanghai”. In our exemplary documentation model we assume this property to be dependent on the actual type, as e.g., not for all application containers the location is known or relevant to be documented. Because of this, we needed to realize a property-like concept, so called *mixin*s [5], that can be instantiated on both, the level of types and documented instances. This is comparable to the concept of *deep instantiation* [1] or that of *intrinsic attributes* in the *MEMO* meta-modelling language [10].

The mixin concept aligns well with the flexible nature of our type concept and allows the documenting stakeholders to adapt the documentation model to cater their particular documentation needs. With mixins we provide a potential solution to challenge **CH4**.

5 Related Work

In the context of EAM it is common that tools provide predefined information models that can often only be adapted in a very limited way. For example, the EAM tool *iteraplan*[3] only allows for the extension of existing classes via attributes. No additional classes or relationships can be added. As shown in the EAM tool survey by Matthes et al.[14] there exist some configurable tools, their technical foundation, however, is not clear. Other tools work with fixed information models based on EA modeling standards such as *The Open Group Architecture Framework* [11] or *Archimate* [13]. We argue that these standards

[3] http://www.iteraplan.de/en

are inflexible as it is difficult to adapt them to the terminology used in an organization or to evolve. Schweda presents a sophisticated approach for pattern-based creation of organization-specific information models [16] and shares our modeling requirements in his research. However, other than the scope of our work, its practical applicability was not shown so far. With the *MEMO* meta-modeling language, Frank et al. [10] present a language and a tool suite for building modeling languages in the enterprise context. The tool is Eclipse-based and needs code generation steps in order to react on a changed information model. The proposed language for IT-infrastructure modeling, ITML [9], provides fixed concepts and can not support organization-specific information models. In line with Kattenstroth [12], we conclude that although the need for organization-specific and evolvable EA information models has been identified in literature [7, 16], related work mostly focuses on formulating generic and fixed information models that cannot be adapted to the requirements of a given organization.

In the general modeling research much related literature can be identified. Still, modeling in this area mainly discusses requirements from software engineering and does not necessarily consider modeling techniques from other domains. For *Txture* we mainly built on top of known modeling paradigms, but unified them in a novel way to contribute a usable EA documentation method. This includes ideas from *UML stereotypes*, *power types*, the proposed separation of *linguistic* and *ontological* models (and instantiations) [2]. Implementation-wise we rely on *Ecore*[4], an object-oriented meta-modeling framework with reflective capabilities and a programming interface for *Java*, which proved to be stable and reliable. For persisting models we used a custom hybrid repository approach including a relational database together with a graph database for permanent storage and fast model query capabilities respectively. These technologies were chosen due to prior experience; another promising alternative to be evaluated for our use case is *MetaDepth* [6], a framework for supporint arbitrary numbers of meta levels and advanced modeling concepts. As we implemented custom form-based modeling user interfaces to ease the IT documentation for non-modelers, we were unable to rely solely on UML and its provided graphical notations; despite many UML concepts are used in our work. *Txture* also consists of a number of different editors for different stakeholder groups and purposes (cf. e.g., the approaches described by Atkinson et al. [3, 4]).

6 Conclusion & Outlook

In this paper we presented the modeling framework *Txture* that offers a flexible mechanism to create organization-specific architecture models. In the main section we described IT-architecture documentation challenges and presented our solutions. These solutions are derived from practical experiences from two industry projects. Based on these experiences we claim that proper architecture modeling requires a hybrid approach, consisting of a classical meta-model hierarchy and multi-level modeling methods. The classical modeling part is important

[4] `http://www.eclipse.org/modeling/emf/?project=emf`

to make documentation capabilities comprehensible to a wide range of different stakeholders in an enterprise. Multi-level modeling, like we employ via types and mixins renders the overall documentation process flexible and extensible. Thus, many of our design decisions were influenced by the overriding principles of practical tool usability and intuitiveness. For example, we favored a free tagging mechanism over type-inheritance.

While we strongly believe that many application domains would benefit from multi-level modeling concepts, however, corresponding modeling frameworks that have proven maturity are rare. This is why we built *Txture* on top of *Ecore*, although we needed to heavily deviate from its intended usage (regarding power types and mixins) to build the here-described framework.

In the future we want to gather additional practical experiences, in order to further evaluate the flexibility of *Txture*'s modeling capabilities.

References

1. Atkinson, C., Kühne, T.: The essence of multilevel metamodeling. The Unified Modeling Language. Modeling Languages, Concepts, and Tools (2001)
2. Atkinson, C., Kühne, T.: Model-driven development: a metamodeling foundation. IEEE Software 20(5) (Sep 2003)
3. Atkinson, C., Gerbig, R.: Harmonizing Textual and Graphical Visualizations of Domain Specific Models Categories and Subject Descriptors. In: Proceedings of the 2nd Workshop on Graphical Modeling Language Development. ACM (2013)
4. Atkinson, C., Gerbig, R., Tunjic, C.: A multi-level modeling environment for SUM-based software engineering. Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling - VAO '13 (2013)
5. Bracha, G., Cook, W.: Mixin-based inheritance. ACM SIGPLAN Notices 25(10), 303–311 (1990)
6. De Lara, J., Guerra, E.: Deep meta-modelling with metadepth. In: Objects, Models, Components, Patterns, pp. 1–20. Springer (2010)
7. Farwick, M., Schweda, C.M., Breu, R., Hanschke, I.: A situational method for semi-automated Enterprise Architecture Documentation. SOSYM (Apr 2014)
8. Farwick, M., Trojer, T., Breu, M., Ginther, S., Kleinlercher, J., Doblander, A.: A Case Study on Textual Enterprise Architecture Modeling. In: Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2013 17th IEEE International. IEEE (2013)
9. Frank, U., Heise, D., Kattenstroth, H., Fergusona, D., Hadarb, E., Waschkec, M.: ITML: A Domain-Specific Modeling Language for Supporting Business Driven IT Management. In: Proceedings of the 9th workshop on domain-specific modeling (DSM). ACM (2009)
10. Frank, U.: The MEMO meta modelling language (MML) and language architecture. 2nd Edition. Tech. rep., Institut für Informatik und Wirtschaftsinformatik (ICB) Universität Duisburg-Essen (2011)
11. Haren, V.: TOGAF Version 9.1. Van Haren Publishing (2011)
12. Kattenstroth, H.: DSMLs for enterprise architecture management. In: Proceedings of the 2012 workshop on Domain-specific modeling (DSM). ACM Press (Oct 2012)
13. Lankhorst, M.: Enterprise Architecture at Work, vol. 36. Springer Berlin Heidelberg, 3rd editio edn. (Jan 2012)
14. Matthes, F., Buckl, S., Leitel, J., Schweda, C.M.: Enterprise Architecture Management Tool Survey 2008. Tech. rep., Technische Universität München,Chair for Informatics 19 (sebis) (2008)
15. Odell, J.J.: Power Types. Journal of OO Programming (1994)
16. Schweda, C.M.: Development of Organization-Specific Enterprise Architecture Modeling Languages Using Building Blocks. Ph.D. thesis, TU Munich (2011)