Taxonomy of Architectural Style Usage

Simon Giesecke
Carl von Ossietzky University of Oldenburg
Software Engineering Group
26129 Oldenburg, Germany
giesecke@informatik.uni-oldenburg.de

ABSTRACT

A taxonomy of architectural style usage is introduced, which serves to design new (agile or heavy-weight) software development methods that employ architectural styles. We use the term "architectural styles" to refer to high-level design patterns. We identified five major usages: ad-hoc, platform-oriented, customized, pre-modeling and post-documentation/-analysis. In addition generic and reference architectures are compared to architectural styles based on their usage. Finally, a classification of these usages is presented that discusses the dimensions compositionality, specialization, explication/rigor, conceptual level, relationship to system quality attributes, and the suitability for architectural design exploration.

1. INTRODUCTION

Reuse has long been an important aspect of software engineering, and it comes in many forms, e.g. code or design reuse [29]. With the rise of software architecture as a discipline [46], reuse has also been elevated to the level of software architecture: Component-based software engineering approaches specifically target the reuse of executable units, i.e. software components. However, interoperability problems may inhibit the composition of components into an architecture [10, 13]. Therefore, other approaches to reuse at the architectural level are required as a prerequisite for component reuse. One way to explicitly represent reusable architectural knowledge are architectural styles. An architectural style can be interpreted as defining a family of software architectures. Their benefits may be exploited by specifically targeted software design methods. In this paper, we propose a taxonomy of the various modes of using architectural style that is meant to support future research into such design methods.

Approach and Contribution.

We developed a taxonomy of architectural styles based on the *usage*, i.e. the mode of use, within software engineering. The taxonomy is not intended to directly support understanding an existing architecture description or designing new systems. It is situated on a meta-level to the design new systems, i.e. to software design *methods*. More specifically, it refers to the development, comparison and discussion of software design methods. Indirectly, however, knowledge on the mode of using architectural styles that has been applied or is intended to be applied helps in understanding the architecture description of a system or in applying some design method in the design of a new system.

The presented taxonomy is more or less indifferent to the form of *representation* of architectural styles (e.g., informally, using an ADL or as graph grammars). However, the specific form of representation may make the style more or less apt to a specific mode of use. Another aspect of styles, which might be used for a taxonomy, is their *genesis* or *origin*. We will only slightly discuss these aspects and focus on the usages. This approach will not yield a classification scheme for architectural styles themselves since any particular style may, in principle, be used in various modes. However, *typical* representatives of styles used in each of the modes can be identified.

In our opinion, the work on architectural styles has been focused too exclusively on aspects of representation and on particular styles, while the use of these styles has been considered only implicitly. Our taxonomy contributes to present knowledge in three ways:

1. From a more theoretical point of view, the usages of styles can be better understood by making them explicit.

2. The existing literature on styles usages is assembled, surveyed, and classified with respect to the style usage that is promoted, making the current state of knowledge more accessible.

3. The knowledge on current usages can be exploited in designing new ways of using architectural styles, fixing deficiencies of current usages or combining features of distinct usages.

The latter aspect was our primary motivation for creating the taxonomy, and it lead to the design of the MIDARCH method for integrating and migrating business software systems on the architectural level [21].

We analyzed the literature on architectural styles and patterns and identified several clusters of modes of use. We will refer to these clusters simply as *usages* in the following for better readability. We first describe and compare these

PLoP '06, October 21-23, 2006, Portland, OR, USA

modes of use in a bottom-up manner. As part of this, we provide references to literature that is relevant to each of the modes of use. Furthermore, we provide typical examples of architectural styles in each mode of use. Based on interesting characteristics identified within this phase, we provide a top-down categorization in Section 4.

Based on the study of the available literature, we identified five major modes of using architectural styles: Ad-hoc, Platform-oriented, Customized, Pre-modeling and Documentation and analysis.

Overview.

The rest of the paper is structured as follows: First, our understanding of the concepts of architectural styles and patterns is described in Section 2. In Sections 3.1 to 3.6, each mode of use is discussed in a bottom-up manner. Afterward, Section 4 provides a classification scheme and fits the modes of use in this classification scheme. The paper is concluded in Section 5.

2. ARCHITECTURAL STYLES, PATTERNS AND THEIR USAGE

Architectural knowledge, such as patterns, reference architecture or guidelines, is represented in forms that are usually more human-oriented at least in some part (documentation), while other parts (models, specifications, example implementations, etc.) may be written in a formal language as well. Architectural knowledge may be available only implicitly, or explicitly in either free natural language or in some codified, i.e. standardized explicit, form. Examples of codification templates are the various pattern templates (Coplien form [12], Gang of Four form [15], etc.), as well as architectural style specifications in ADLs such as ACME [17].

We regard the notion of *architectural styles* as one of the most interesting, but perhaps also most often misinterpreted concepts in the area of codified architectural design knowledge.

The idea of architectural styles more or less developed within the ADL community, which follows a more formally-oriented perspective on software architecture. In parallel, the idea of design patterns (in the following, we will use the term "pattern" in brief) evolved and a pattern community grew. This community follows a more pragmatic perspective on software architecture and therefore consists of software development practitioners to a larger degree. The ADL community, on the other hand, is dominated by academic and industrial software engineering researchers.

Perhaps due to their pragmatic perspective, some proponents of the pattern community claim that styles may be subsumed by the idea of patterns [8]. This is true to some degree, as already noted in [39], but captures only a specific usage and requires a very broad interpretation of the pattern concept. Intriguingly, on the one hand, the pattern community seems to have very specific ideas what patterns are and what role they play in software development, but on the other hand they are unable to give a rigorous definition of a pattern, which would enable deciding if a given artifact is a pattern or not [48]. The discussions reduce to the question of syntactic representation and relationships of patterns. These paths do not really help in deciding which artifacts are patterns, and which are not: If the descriptions

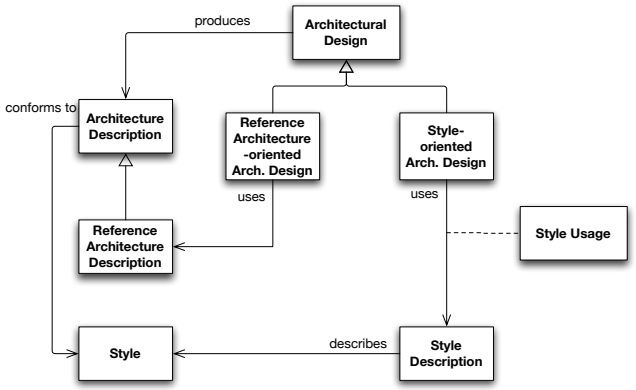

Figure 1: Major Modes of Using Architectural Styles

contain natural language, which is an essential part of the pattern descriptions in many cases, and merely a coarse-grained structure for the text is prescribed, arbitrary concepts may be put into the form of a pattern. Thus, the form cannot suffice for deciding whether something is a pattern or not. There has been some work on the formalization of pattern descriptions [41], but this is somewhat decoupled from the mainstream pattern community.

Since architectural styles and patterns are not usually distinguished consistently, we considered both as nearly equivalent during creating the taxonomy. In this paper, we will use the terms "design patterns" for lower-level artifacts, and "architectural styles" for higher-level artifacts, except when explicitly discussing differences between both concepts. A taxonomy of these concepts and other types of architectural constraints is presented in [23].

Figure 1 shows an overview of our view of the relationship of architectural styles to other concepts of software architecture, which applies to all the usages we identified. For simplicity, we have neglected multiplicities of the associations in the diagram. Style-based architectural design is a special kind of architectural design, which uses style descriptions in a way determined by the style usage. A style description is an explicit representation of a style. Any architecture description conforms to some style; whether it explicitly references a style description is left open in this diagram.

3. TAXONOMY

Based on the study of the available literature, we identified five major modes of using architectural styles, which are shown in Figure 2. Also mentioned in the diagram is the MIDARCH Usage of architectural styles, which is an example of a specific design method exploiting architectural styles [20, 22].

Additionally, we discuss the relationship of these usages to the use of generic or reference architectures. These are units of architectural knowledge which are not exactly architectural styles, but similar artifacts.

3.1 Ad-hoc Use of Styles

As already mentioned above, sometimes the style concept is subsumed under the pattern concept (see, e.g., [8,26,44]). In this view, every style may be expressed as a design pattern, but not necessarily vice versa. This view may also

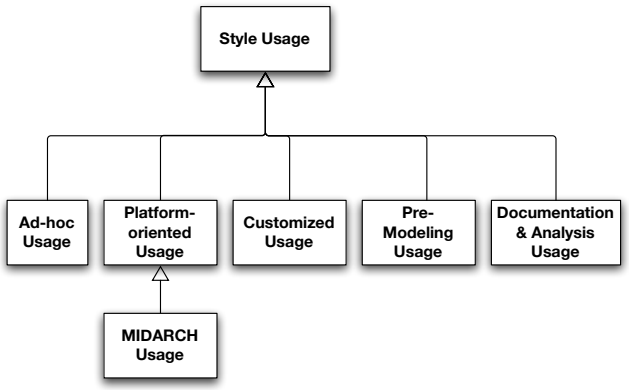

Figure 2: Major Modes of Using Architectural Styles

be referred to as the *styles-as-patterns view*. These authors conceive to an essentially ad-hoc usage of styles: The consultation of a pattern collection and the selection of a pattern to apply remains at the discretion of the architects on the basis of their experience. Collections of patterns specify different kinds of relationships between multiple patterns (see, e.g., [50]), but the relationships that refer to their usage cannot be made explicit in general. Thus, using patterns/styles in this way is difficult to teach or convey.

In principle, everything that has been said about using patterns also applies to styles in this view [42, 43]. However, since not every design pattern is viewed as a style even by the proponents of this view, some additional remarks regarding the use of "typical" styles-as-patterns are necessary. The relatively simplistic general-purpose styles originally described in [46], for example, are criticized as being impossible to apply to a system as a whole [45]. Indeed, this is a reflection and manifestation of the ad-hoc approach to using styles-as-patterns: styles are not assumed to guide the following architectural development, but they are reduced to communication styles between two or more system elements. For any two elements that should communicate with each other, a different communication style might be chosen. This contradicts the original view of a style as embodying a *decomposition principle*: the idea was that different styles led to different system decompositions [18]. On the contrary, the styles-as-patterns view tends to view the decomposition of a software system to be relatively independent from the styles employed and the style just influences the interaction between pre-established, i.e. established prior to the selection of the style, elements. While this view masks several aspects of architectural styles deemed relevant by other authors, such as providing style-specific analyses, the scenario is of major practical relevance when re-engineering an existing system or building a new system of existing (COTS or not) components. The current state of the architecture may be very heterogeneous, which is the ultimate reason for re-engineering it. Still, it may be infeasible as well to provide a coherent target architecture which follows a single architectural style, e.g. because some components are considered intangible because they embody knowledge that cannot be safely restored. It is in principle always possible to define an overlay architecture on top of the implemented architecture that is coherent, but this may not always be worth the effort.

Relationship to other modes of use.

This usage should not be confused with the usage that regards a style as defining a pattern *language* (cf. [39]), which again puts styles at a level above patterns. This view is discussed in Section 3.3.

Some "patterns" are actually complete, though very abstract, generic architectures (e.g., the Model-View-Controller architectural pattern) rather than patterns. They are discussed in Section 3.6.

When re-engineering a software system, the first step may be the re-documentation of the current architecture (see Documentation and analysis, Section 3.5). For the reasons discussed above, often the only way to continue using styles is then an ad-hoc usage.

Typical examples.

Typical examples can be found in the extensive literature on architectural patterns. Well-known are the POSA books of Buschmann and others, the first of which names the Layers, Pipes and Filters, Blackboard, Broker, Model-View-Controller (MVC), Presentation-Abstraction-Control (PAC), Microkernel and Reflection architectural patterns.

3.2 Use as Platform-oriented Styles

At first sight, this usage appears to be tightly bound to the genesis of the architectural styles, namely their derivation from the platform that shall be used. The term "platform" is used in a very generic sense here. The discussion of what an (abstract) platform is in the sense of the MDA is related (cf., e.g., [4]). For our purposes, target or *implementation* platforms must be distinguished from *modeling* platforms. A modeling platform is a modeling language or notation, for example an ADL. There are ADLs that are specifically designed to support a specific architectural style, e.g. C2SADL [33] for the C2 style [47], but even ADLs that claim to be style-independent are usually biased towards some architectural styles [14]. Implementation platforms can be distinguished into several types again:

System Software The system software (operating system) and the services it offers to applications may impose an architectural style. However, in particular for distributed applications the actual operating system is today often hidden behind an additional software layer referred to as middleware.

Middleware A middleware layer is some software layer that is found between the operating system and application layer. Its particular characteristics depend on the type of application that is considered. Middleware-induced architectural style have received some attention from the research community [6, 14, 31, 32]. They are also in the focus of our main research [19].

Programming Paradigm The *language constructs* (classes, objects, exception, event mechanisms, etc.) a programming language provides and its implied *execution model* advocate some programming paradigm. For example, the execution model of languages such as Java and C++ promotes the synchronous communication model between objects. The paradigm of the language(s) considered for implementing a system may

influence the structure of the resulting system at an architectural level [34]. For relatively low-level languages, such as C, only very few constraints are implied, but for very high-level languages, a very specific architectural style may result, in particular if the language effectively inhibits access to lower-level constructs.

Hardware Besides the software artifacts mentioned before, the hardware (or an abstract, virtual machine) also imposes some style on the software system, if only indirectly through the software layers in between. For distributed software systems, however, the physical topology and the properties of physical connections inevitably influence quality properties of the running system, and must thus be considered in system design.

Organizational Structure The famous Conway's law already postulated the dependence of system structure on the structure of the designing or developing organization: "[...] organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations" [11]. While the inevitability of this constraint may be questioned, it is clear that a tendency towards it prevails and it may adversely affect the structure of the system from a technical point of view without compensating measures. Similarly, the organizational structure of the target organization may influence the architecture.

All in all, a software system makes use of many platforms that may suggest different architectural styles. The architecture may be specified independently from the platforms it is implemented on. However, a trade-off must be made between the platform-independence and the achievable quality of the mapping to the target platform. One quality characteristic of the mapping is its complexity, which should be as low as possible. Furthermore, the risk that the mapping will lead to a system with undesired quality characteristics should be reduced. Such a risk may arise from incongruities between the structure implied by the architecture and the structural characteristics that are beneficial to the platform.

It is possible to define a taxonomy of platforms based on the architectural styles they enable [14]. In defining such a taxonomy, abstract platforms may need to be introduced in addition to physically existing platforms, which reliefs the tight binding of this usage to the genesis of the style. Based on a taxonomy, the style may be incrementally refined from an abstract towards a concrete style in a process based on this usage, yielding a refinement hierarchy of architectures as well. Unfortunately, only few taxonomies of platform-oriented styles are available yet [14, 22, 31, 32].

Typical examples.

Di Nitto et al. consider the styles induced by the (event-based) JEDI and C2 middleware technologies. C2 is also discussed by Medvidovic et al., as well as its relationship to COM, CORBA, and Java RMI. In our own work, we currently focus on the styles imposed by the Apache Merlin and Apache Cocoon middleware frameworks.

3.3 Use as Customized Styles

This is the primary usage intended by the community which focuses on ADLs that support the definition of architectural styles and conforming architectures, e.g. Rapide [30], Acme [17] resp. Armani [38], Wright [3], ArchWare [5, 40], and Alfa [35, 36]. There has been important work on formalizing the properties of architectural styles using specifications in the general-purpose specification language Z [1].

Garlan [16] discusses several different approaches to the definition and use of architectural styles, but assumes several common properties of any definition of the architectural style concept:

a. The provision of a "vocabulary of design elements", which are "component and connector types".

b. The definition of a "set of configuration rules".

c. The definition of a "semantic interpretation", which gives some well-defined meaning to all configurations of design elements that satisfy the configuration rules.

d. The definition of "analyses" for configurations of that style. Examples include schedulability analysis, deadlock analysis, code generation, and conformance checking.

Styles used in this way may be seen as a concept complementary to patterns according to Monroe et al. [39], who point out that "for a given style there may exist a set of idiomatic uses. These idioms act as microarchitectures, or architectural design patterns, designed to work within a specific architectural style." The architectural style provides general guidance for the architectural development process, while architecture-level design patterns solve specific problems within one or multiple styles [39]. As an example, Monroe et al. give the Real-time Producer/Consumer style with two subordinate architectural design patterns, the Shared-resource and Message-Replications patterns.

Monroe et al. also point out that an architectural style can be regarded not merely as a form of pattern, but as defining a whole pattern language. However, the term "pattern language" is controversial in itself. It is misleading as it may be thought to appeal to the common notion of a formal language, at least for computer scientists. It was used in [2] as a vague metaphor to natural languages, but the same concept could better be referred to as a pattern vocabulary (cf. [43]).

Typical examples.

Abowd et al. [1] discuss different variants of the Pipe-and-Filter and Event-based architectural styles that are tailored to specific needs.

Additional relevant literature.

Keshav et al. [25] propose a taxonomy of architectural integration strategies, among which are both patterns and styles. While the basic elements they identify—Translators, Controllers, Extenders, and combinations thereof—are more or less patterns or individual components. In addition, they identify "loosely defined integration strategies", which do not fit into their main taxonomy, but may be regarded as architectural styles.

3.4 Style-based Pre-modeling

This usage is tightly bound to a specific type of architectural styles, which is briefly introduced in the following.

Klein et al. [27, 28] provide a derivative of the original architectural style concept (as described in Section 3.3) which focuses more explicitly on the quality characteristics of the resulting architectures. These are referred to as "Attribute-based Architectural Styles" (ABAS). One ABAS is assumed to *address* one quality attribute[1]. A system is modeled according to several ABASs, addressing the most critical attributes, yielding several architectural models. Since each model strictly conforms to a style, style-based analyses (cf. Section 3.3) may be performed on them. When each model is satisfactory in terms of its assigned quality attribute, an overall system model is synthesized from the individual models.

By this approach, the original idea of different styles leading to different system decompositions is combined with (some limited form of) heterogeneity of architectural styles within a system. However, the way the final system is derived from the multiple decompositions is not specified (which is probably not possible in the generality that the approach targets). Since it is known that many quality attributes require trade-offs, the properties of the resulting architecture may not conform to the properties of the individual style-based models. Furthermore, it is unclear where later changes to the architecture should be performed.

This usage of styles may be referred to as "style and forget", since after the style-based modeling has been done initially the benefits of architectural styles are abandoned again. The concept of ABASs, however, seems promising and might be used in the opposite way, which is described in Section 3.5.

Typical examples.

Example ABASs are the Synchronization ABAS for communicating processes exchanging streams, which allows latency analysis; and the Data Indirection ABAS, which allows analyses of the degree of coupling. The latter style provides the substyles of an Abstract Data Repository and the Publish/Subscribe ABAS.

3.5 Style-based Documentation and Analysis

This usage may be regarded the reverse of the previously described usage. Styles are not considered in the primary development of the architecture, but only applied to intermediate or final results of defining the architecture. This mode of use is discussed in [9], for example. They may serve to document the architecture by interpreting it in terms of a specific architectural style, to ease understanding of the overall system. Additionally, style-specific analyses may be applied to the style-based views. An automatic mapping of a generic architecture to a style will not be possible, but the mapping may be defined once and needs only be adapted, when the generic architecture changes. Perhaps even dependent refactorings, which are attached to refactorings on the primary system architecture, might be defined.

Essentially, any approach to software architecture that defines multiple views on the architecture may be interpreted as this usage. Similarly to generic ADLs, architectural views impose some style upon the modeled entity, but usually this style is conceptually not very deep.

[1]in the ISO 9126 terminology, it is probably more appropriate to speak of a quality (sub-)characteristic

Typical examples.

As discussed above, the same architectural styles as considered in pre-modeling can be used for post-modeling and documentation. In addition, Clements et al. discuss architectural styles for the following viewtypes: The Decomposition, Uses, Generalization, and Layered styles for the Module Viewtype; the Pipe-and-Filter, Shared-Data, Publish-Subscribe, Client-Server, Peer-to-Peer, and Communicating-Processes styles for the Components & Connectors Viewtype; and the Deployment, Implementation and Work Assignment styles for the Allocation Viewtype.

3.6 Use as Generic Architectures

This usage is not strictly a usage of architectural styles, since the artifacts that are employed in this usage do not fit the definition of architectural styles, but represent a different type of architectural constraint, namely generic architectures. A generic architecture is expressed within the same (or similar) language or notation as a (product) software architecture, but may employ a different interpretation. A generic architecture may be regarded as a template that is enriched or refined into a product software architecture. In some cases, a generic architecture can also be interpreted immediately as the architecture of a product system.

Examples of architectural styles that are essentially generic architectures are the Model-View-Controller architecture and N-tier architectures (for a given N). These generic architectures can themselves be explicitly based on an architectural style in another usage. N-tier architectures, for example, can obviously be regarded as following the Layered style.

Related Concepts.

Reference architectures are usually generic architectures (see, e.g., [24]). Some approaches to product-line architectures view a product-line architecture as a special kind of reference architecture that is specific to the product line. Such product-line architectures may also be regarded as generic architectures in our sense. More elaborate product-line architecture approaches, which automatically derive individual product architectures or implementations, are beyond the scope of our considerations.

Relationship to other modes of use.

This usage is sometimes regarded a special case of the ad-hoc use, when only one pattern is used and this pattern has system-wide scope.

4. CHARACTERISTICS OF THE MODES OF USE

After the individual modes of use have been identified and described, we now devise a set of characteristics and classify each of the modes of use with respect to these characteristics. The characteristics describe several requirements on the form of representation of the respective architectural styles. We thereby establish the link to the typical level of discussion of architectural styles.

We consider the following characteristics:

Compositionality The characteristic "Compositionality" describes whether the composition of individual styles into a new style is possible, either manually or automatically.

Specialization The characteristic "Specialization" describes whether specialization relationships between styles are relevant within the respective usage.

Explication and Rigor The characteristic "Explication and Rigor" describes whether the architectural styles considered in the respective usage need to be explicitly described in what detail and with what degree of formal rigor.

Conceptual Level The characteristic "Conceptual level" describes whether the architectural styles considered in this mode of use are (typically) bound to technical concepts (of an existing or planned technical platform) or if they are more abstract in nature.

Relationship to System Quality Characteristics This characteristic describes which kind of relationship the styles exhibit to the quality characteristics of the system. Of course, an architectural style always has some implicit relationship to the system quality, but we consider only explicated relationships here. In our view, system quality comprises external and internal implementation quality as well as architectural quality (cf. realms of software quality discussed in [7]). Essentially two types of relationships can be considered here: An explicit reference to some system quality characteristic that a style addresses, and the reference to analysis techniques that are enabled by the style (cf. [16]).

Suitability for Architectural Design Exploration Architectural design exploration is a design activity which covers the systematic specification of multiple candidate architectures and their evaluation with respect to architectural and system quality characteristics. This characteristic describes which role the respective usage of styles can play in the architectural design exploration process, i.e. how either different styles or different design based on a style can be evaluated in that usage. Since the modes of use do not exclude each other, this only refers to the contribution of the considered usage.

An overview of the classification is given in Table 1, the details are explained for each characteristic in the following subsections.

4.1 Compositionality

Let S be the set of architectural styles and A the set of architectures. Style composition may be performed either at the type or instance level.

Style composability at the *instance level* means the following: Let there be two styles $x, y \in S$. If an architecture $a \in A$ conforms to a style x, it is modified in a way that it conforms to style x and conforms to style y as well. Then x and y are composable for a. Thus, there is a (partial) composition operator $\bar{\times}_1 : (A \times S) \times S \to A$ such that:

$$(\bar{\times}_1(a, x, y) = b \Rightarrow b \text{ conforms to } y) \quad (1)$$

$$\wedge\, (\bar{\times}_1(a, x, y) \text{ defined} \Rightarrow a \text{ conforms to } x) \quad (2)$$

Style composability at the *type level* is in place if a (partial) composition operator $\bar{\times}_2 : S \times S \to S$ is defined on the set of architectural styles. Composability at the type level does not imply that two architectures $a, b \in A$ in two different styles $x, y \in S$ can be meaningfully combined into a new architecture conforming to $a\ \bar{\times}_2\ b$.

Compositionality can either be impossible resp. *undefined*, *manual*, *semi-automatic* or *automatic*. It is automatic if the composition operator is computable. If the composition operator is only partially computable, or if the determination of the composition is partially supported by an program, compositionality is considered semi-automatic. Of course, automatic compositionality is only conceivable for styles which are formally specified.

Classification.

The composition of styles used in an AD-HOC manner is expressly *manual* at the *type level*, and *semi-automatic* on the *instance level*. On the type level, new styles may be designed by combining the ideas underlying existing styles, which is a creative process for the most part. On the instance level, composition is often necessary, as no single style is supposed to be apt to support the development of a whole architecture. Tools might support the instantiation of new styles into an existing architecture description, but conflicts may occur, which cannot be resolved automatically.

In general, the composition of PLATFORM-ORIENTED styles is considered to be essentially *undefined*. In special cases, composition may be simulated through multiple inheritance of platform-oriented styles [22].

Compositionality at the instance level is not applicable in DOCUMENTATION usage, since the derived views are not meant to be tangible.

4.2 Specialization

Specialization relationships between different styles are conceivable for all of the modes of use, but play a different role for each of them. Specialization relationships can be distinguished into single and multiple inheritance. Whether subtyping concepts from type theory can be applied depends on the degree of formality of style specification.

Classification.

In the case of AD-HOC usage, specialization is one of many relationships that are defined between some patterns. It might be exploited in choosing a pattern fitting a problem at hand in a stepwise process, but no work on such a process has been published. Since specialization relationships are typically specified informally, multiple inheritance is possible without introducing additional problems.

For PLATFORM-ORIENTED styles, it is possible to exploit specialization relationships in the development process. The style may be incrementally refined from an abstract style towards a concrete style, yielding a parallel hierarchy of middleware platforms. The resulting hierarchy may also make use of multiple inheritance [19].

When defining CUSTOMIZED styles, it is efficient to reuse existing style definitions by defining the new style as a specialization, which also allow the reuse of analysis and design tools existing for that style. The same applies to style-based pre-modeling and documentation.

4.3 Explication and Rigor

In principle, the level of explication and rigor is independent from the mode of use. Still, certain *minimal* levels that the style specification must fulfill can be determined on the one hand, and *typical* levels that can be found can be

	Ad-hoc	Platform-oriented	Customized	Pre-modeling	Documentation
Compositionality	Manual	No	Manual ?	No	n.a.
Specialization	Yes	Yes	Yes	Yes	Yes
Explication & Rigor	Informal	No	Formal	Formal	Informal
Conceptual Level	(Abstract)/Technical	Technical	Abstract/(Technical)	Abstract	Both
Relationship to System Quality Characteristics	No	Analyses/Empirical	Analyses	(Analyses)	Analyses
Suitability for Architectural Design Exploration	Limited	Yes	Partially	Yes	No

Table 1: Characteristics of the Architectural Style Usages

identified as well.

Classification.

In an AD-HOC use of styles, the level of rigor is typically informal. Architectural patterns are only described very vaguely by just giving examples of their instances that do not claim generality at all.

PLATFORM-ORIENTED styles are seldom described explicitly at all yet, with the exception documented in [14]. However, the great variety of existing middleware platforms has not yet been specified formally. In fact, access to the style description might not be necessary at all after a taxonomy of platforms has been derived based on formal style modeling. Exploitation of additional style features still requires the explicit use of formal style specifications.

In CUSTOMIZED usage, formal specification of styles is expected in the context of ADL-based architecture specification. Similarly, since formal analyses are an important aspect in style-based PRE-MODELING, thus a formal definition of the style is required.

The required level of rigor depends on the focus in DOCUMENTATION usage. If the focus is on documentation, informal and vague specification of the style may suffice, but when formal analyses should be performed, formal style specification is necessary as well.

4.4 Conceptual Level

The characteristic "Conceptual level" describes whether the architectural styles considered in this usage are (typically) bound to technical concepts (of an existing or planned technical platform) or if they are more abstract in nature. This distinction corresponds to different conceptual levels at which software architecture can be specified in general (e.g. conceptual vs. concrete architectures [37] or physical vs. logical architecture [49]).

Classification.

For AD-HOC usage, both types are possible. While the old general-purpose architectural styles [18] are quite generic and abstract, current publications on architectural patterns are focused towards specific platforms, and are either specifically tailored towards one platform or provide examples for multiple platforms and are thus more technically oriented.

Naturally, PLATFORM-ORIENTED styles are conceptually close to the platform they intend to support and are thus technical in nature. However, generalizations made in a taxonomy of platform styles may introduce concepts that have no direct counterparts in any existing platform.

CUSTOMIZED styles can be both abstract and technical in nature. Due to the fact that the ADLs used for specification are conceptually quite detached from typical implementation techniques, abstract styles are more prevalent.

In PRE-MODELING usage, architectural styles are used in an early stage of architectural development only, before the final architecture of the system to be built is established. Since the link of the pre-models and the final architecture is quite loose, the styles in this usage are very abstract.

Finally, for DOCUMENTATION usage, both types of styles are conceivable.

4.5 Relationship to System Quality Characteristics

Architectural styles are intended to improve the process quality of software development by acting as an intellectual tool to the system designers. Additionally, they are intended to improve the product quality of the product that is the outcome of the development process. The different modes of use support the product quality improvement in differing ways.

Classification.

Due to the vague nature of the styles used AD-HOC, a relationship to system quality characteristics is difficult to establish. The architectural pattern literature lists experience-based hints on when to use which patterns, but the general applicability of these rules is questionable.

Through defining multiple PLATFORM-ORIENTED styles in a commensurable way, styles and systems based on these styles can be analysed and empirically evaluated to produce a guideline for choosing a suitable style and platform for a given scenario.

Styles in CUSTOMIZED as well as DOCUMENTATION usage often allow style-specific analyses [16]. In PRE-MODELING, these analyses only establish properties of the pre-models, and the link to the actual system's quality is unknown.

4.6 Suitability for Architectural Design Exploration

Architectural design exploration aims to support making trade-offs between system quality characteristics in the development process. This characteristic serves as a summary judgment and combines the previous characteristic with the possibility to exploit the relationship to the quality characteristics in the architectural development process.

Classification.

Due to the AD-HOC nature of using styles, the suitability

for systematic architectural design exploration is very limited.

For PLATFORM-ORIENTED style usage, a design method is developed [21] that is targeted at supporting the choice of an appropriate style and platform through tools and the provision of guidelines.

Using styles as CUSTOMIZED styles is only partially suited for architectural design exploration. Many choices must be made in customizing the style, i.e. before the actual modeling is done. A method for architectural design exploration on this level is conceivable, but has not yet been proposed and evaluated in detail.

PRE-MODELING usage is in fact the most direct correspondence of the idea of architectural design exploration. The main issue with this usage is the linking of the architectural design exploration with the actual modeling.

DOCUMENTATION has a post-mortem relationship to modeling activities, i.e. a modeling phase has been completed when the documentation is consolidated. Thus, the suitability for architectural design exploration is very limited. Perhaps well-integrated tool support that enables a frequent automatic derivation of style-based views would allow for enabling architectural design exploration.

5. CONCLUSION

A taxonomy of architectural style usages is presented. We identified five major modes of using architectural styles, which are discussed in detail, based on the relevant literature. In addition, while showing some commonalities with architectural styles, generic or reference architectures have been distinguished from architectural styles based on their usage. Besides this bottom-up-oriented discussion, a top-down classification of the identified usages discussing compositionality, specialization, explication/rigor, conceptual level, relationship to system quality characteristics, and the suitability for architectural design exploration has been presented.

In our work, this coarse-grained taxonomy is currently used to further explore platform-oriented usages of architectural styles, especially where the platforms to which the styles are oriented are middleware platforms in the widest sense. The MIDARCH Design Method exploits architectural styles for design guidance and knowledge transfer in migration and integration projects [19, 21, 22]. This work will benefit from the research results presented in this paper, because it provides the basis for incorporating beneficial features from other modes of use. Similar work on other modes of using architectural styles may be inspired by our work as well.

6. REFERENCES

[1] G. D. Abowd, R. Allen, and D. Garlan. Formalizing style to understand descriptions of software architecture. *ACM Trans. Softw. Eng. Methodol.*, 4(4):319–364, 1995.

[2] C. Alexander, S. Ishikawa, and M. Silverstein. *A pattern language : towns, buildings, construction*, volume 2 of *Center for Environmental Structure*. Oxford Univ. Press, New York, 1977.

[3] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 6(3):213–249, 1997.

[4] J. P. Almeida, R. Dijkman, M. van Sinderen, and L. F. Pires. On the notion of abstract platform in mda development. In *EDOC '04: Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International (EDOC'04)*, pages 253–263. IEEE Computer Society Press, 2004.

[5] D. Balasubramaniam, R. Morrison, G. N. C. Kirby, K. Mickan, and S. Norcross. Archware adl release 1 user reference manual. Technical Report D4.3, ArchWare Project, 2004.

[6] L. Baresi, R. Heckel, S. Thöne, and D. Varró. Style-based refinement of dynamic software architectures. In *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA'04)*, page 155. IEEE Computer Society, 2004.

[7] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Longman, 1. edition, 1998.

[8] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 1996.

[9] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little. *Documenting Software Architectures: Views and Beyond*. Pearson Education, 2002.

[10] D. Compare, P. Inverardi, and A. Wolf. Uncovering architectural mismatch in component behavior. *Science of Computer Programming*, 33(2):101–131, 1999.

[11] M. E. Conway. How do committees invent? *Datamation*, 14(4):28–31, Apr. 1968.

[12] J. Coplien. *Software Patterns*. SIGS, New York, 1996.

[13] L. Davis, R. F. Gamble, and J. Payton. The impact of component architectures on interoperability. *J. Syst. Softw.*, 61(1):31–45, 2002.

[14] E. Di Nitto and D. Rosenblum. Exploiting ADLs to specify architectural styles induced by middleware infrastructures. In *Proceedings of the 21st international conference on Software engineering*, pages 13–22. IEEE Computer Society Press, 1999.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[16] D. Garlan. What is style? In D. Garlan, editor, *Software architectures*, volume 106 of *Dagstuhl-Seminar-Report*, Saarbrücken, Germany, February 1995. Proceedings of the Dagstuhl Workshop on Software Architecture.

[17] D. Garlan, R. T. Monroe, and D. Wile. Acme: architectural description of component-based systems. In *Foundations of component-based systems*, pages 47–67. Cambridge University Press, New York, NY, USA, 2000.

[18] D. Garlan and M. Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, pages 1–39, Singapore, 1993. World Scientific Publishing Company.

[19] S. Giesecke. A method for integrating enterprise information systems based on middleware styles. In

G. A. Papadopoulos and J. Filipe, editors, *International Conference on Enterprise Information Systems (ICEIS'06) Doctoral Symposium, Paphos, Cyprus*, pages 24–37. INSTICC Press, Portugal, 2006.

[20] S. Giesecke. Middleware-induced styles for enterprise application integration. In *Proc. 10th European Conference on Software Maintenance and Reengineering (CSMR06), Bari, Italy*, pages 334–340. IEEE Comp. Soc., 2006.

[21] S. Giesecke and J. Bornhold. Style-based architectural analysis for migrating a web-based regional trade information system. In A. Trentini, A. Marchetto, and C. Bellettini, editors, *First International Workshop on Web Maintenance and Reengineering (WMR 2006)*, volume 193 of *CEUR Workshop Proceedings*, pages 15–23, 2006.

[22] S. Giesecke, J. Bornhold, and W. Hasselbring. Middleware-induced architectural style modelling for architecture exploration. In *Working IEEE/IFIP Conference on Software Architecture (WICSA 2007), January 2007, Mumbai, India*. IEEE Computer Society Press, 2007.

[23] S. Giesecke, W. Hasselbring, and M. Riebisch. Classifying architectural constraints as a basis for software quality assessment. *Advanced Engineering Informatics*, 21(2):169–179, Apr. 2007. Special Issue on Ontology and Epistemology of Systems and Software Engineering.

[24] A. Grosskurth and M. W. Godfrey. A reference architecture for web browsers. In *ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*, pages 661–664, Washington, DC, USA, 2005. IEEE Computer Society.

[25] R. Keshav and R. Gamble. Towards a taxonomy of architecture integration strategies. In *ISAW '98: Proceedings of the third international workshop on Software architecture*, pages 89–92, New York, NY, USA, 1998. ACM Press.

[26] M. Kirchner and P. Jain. *Pattern-oriented software architecture, vol. 3: Patterns for resource management*. Wiley series in software design patterns. Wiley, Chichester, 2004.

[27] M. Klein and R. Kazman. Attribute-based architectural styles. Technical Report CMU/SEI-99-TR-022, Software Engineering Institute, Carnegie Mellon University, 1999.

[28] M. Klein, R. Kazman, and R. Nord. A basis (or abass) for reasoning about software architectures. Software Engineering Institute, 2000.

[29] C. W. Krueger. Software reuse. *ACM Comput. Surv.*, 24(2):131–183, 1992.

[30] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann. Specification and analysis of system architecture using Rapide. *IEEE Trans. Softw. Eng.*, 21(4):336–355, 1995.

[31] N. Medvidovic. On the role of middleware in architecture-based software development. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 299–306. ACM Press, 2002.

[32] N. Medvidovic, E. M. Dashofy, and R. N. Taylor. The role of middleware in architecture-based software development. *International Journal of Software Engineering and Knowledge Engineering*, 13(4):367–393, 2003.

[33] N. Medvidovic, D. S. Rosenblum, and R. N. Taylor. A language and environment for architecture-based software development and evolution. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 44–53, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[34] N. Mehta, N. Medvidoviç, and M. Rakiç. Why consider implementation-level decisions in software architectures? Technical Report USC-CSE-2000-500, University of Southern California, Computer Science Department, 2000.

[35] N. R. Mehta and N. Medvidovic. Distilling software architecture primitives from architectural styles. Technical Report USC-CSE-2002-509, University of Southern California, Computer Science Department, 2002.

[36] N. R. Mehta and N. Medvidovic. Composing architectural styles from architectural primitives. In *Proceedings of the 9th European software engineering conference held jointly with 10th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 347–350. ACM Press, 2003.

[37] N. R. Mehta, N. Medvidovic, and S. Phadke. Towards a taxonomy of software connectors. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 178–187, New York, NY, USA, 2000. ACM Press.

[38] R. T. Monroe. Capturing software architecture design expertise with armani. Technical Report CMU-CS-98-163, Carnegie Mellon University, School of Computer Science, Sept. 2000. Version 2.3.

[39] R. T. Monroe, A. Kompanek, R. Melton, and D. Garlan. Architectural styles, design patterns, and objects. *IEEE Software*, 14(1):43–52, Jan. 1997.

[40] F. Oquendo, B. Warboys, R. Morrison, R. Dindeleux, F. Gallo, H. Garavel, and C. Occhipinti. Archware: Architecting evolvable software. In F. Oquendo, B. Warboys, and R. Morrison, editors, *Software Architecture, First European Workshop, EWSA 2004, St Andrews, UK, May 21-22, 2004, Proceedings*, volume 3047 of *Lecture Notes in Computer Science*, pages 257–271. Springer, 2004.

[41] R. R. Raje and S. Chinnasamy. elelepus – a language for specification of software design patterns. In *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing*, pages 600–604, New York, NY, USA, 2001. ACM Press.

[42] D. Riehle and H. Züllighoven. Understanding and using patterns in software development. *Theory and Practice of Object Systems*, 2(1):3–13, 1996.

[43] D. C. Schmidt, R. E. Johnson, and M. Fayad. Software patterns. *Communications of the ACM*, 39(10):37–39, Oct. 1996. Special Issue on Patterns and Pattern Languages.

[44] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-oriented software architecture, vol. 2: Patterns for concurrent and networked objects*. Wiley series in software design patterns. Wiley, Chichester, 2000.

[45] M. Shaw. Architectural issues in software reuse: it's not just the functionality, it's the packaging. *SIGSOFT Softw. Eng. Notes*, 20(SI):3–6, 1995.

[46] M. Shaw and D. Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., 1996.

[47] R. N. Taylor, N. Medvidovic, K. M. Anderson, J. E. James Whitehead, J. E. Robbins, K. A. Nies, P. Oreizy, and D. L. Dubrow. A component- and message-based architectural style for GUI software. *IEEE Trans. Softw. Eng.*, 22(6):390–406, 1996.

[48] P. van Emde Boas. Resistance is futile; formal linguistic observations on design patterns. Technical Report ILLC-CT-97-02, ILLC, FWINS, Universiteit van Amsterdam, Feb. 1997.

[49] A. Zendler and H. G. Schwartzel. From logical to physical software architectures. *IETE technical review*, 15(5):355–369, 1998.

[50] W. Zimmer. Relationships between design patterns. In J. O. Coplien and D. C. Schmidt, editors, *Pattern languages of program design*, pages 345–364, Reading, 1995. Addison-Wesley. Proc. PLoP'94.