Taxonomy of Architectural Style Usage

Simon Giesecke*, Wilhelm Hasselbring
Carl von Ossietzky University of Oldenburg
Department of Computing Science
Software Engineering Group
26111 Oldenburg, Germany
{giesecke,hasselbring}@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 (KRUEGER, 1992). With the rise of software architecture as a discipline (SHAW and GARLAN, 1996), 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 (COMPARE et al., 1999; DAVIS et al., 2002). 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

*This work has been partially supported by the German Research Foundation (DFG), grant GRK 1076/1.
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 rep-
resentation 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 (Giesecke and Bornhold, 2006).

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 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 (Coplien, 1996), Gang of Four form (Gamma et
al., 1995), etc.), as well as architectural style specifications in ADLs such as ACME (Garlan et
al., 2000).

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 (Buschmann et al., 1996). This is true to some degree, as already noted by Monroe et al. (1997), 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 (van Emde Boas, 1997). 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 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 (Raje and Chinnasamy, 2001), 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 Giesecke et al. (2006).

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 (Giesecke, 2006a; Giesecke et al., 2007).

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., Buschmann et al., 1996; Kirchner and Jain, 2004; Schmidt et al., 2000). In this view, every style may be expressed as a design pattern, but not necessarily vice versa. This view may also 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., Zimmer (1995)), 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 (Riehle and Züllighoven, 1996; Schmidt et al., 1996). 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 by Shaw and Garlan (1996), for example, are criticized as being impossible to apply to a system as a whole (Shaw, 1995). 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 (Garlan and Shaw, 1993). 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

Figure 2: Major Modes of Using Architectural Styles
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. MONROE et al., 1997), 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., ALMEIDA et al. (2004)). 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 (MEDVIDOVIC et al., 1999) for the C2 style (TAYLOR et al., 1996), but even ADLs
that claim to be style-independent are usually biased towards some architectural styles (DI NITTO
and ROSENBLUM, 1999). 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 sys-
tem 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 (BARESI et al., 2004; MEDVIDOVIC, 2002; MEDVIDOVIC et al., 2003;
DI NITTO and ROSENBLUM, 1999). They are also in the focus of our main research (GIESECKE,
2006b).

Programming Paradigm The language constructs (classes, objects, exception, event mechanisms,
etc.) a programming language provides and its implied execution model advocate some pro-
gramming paradigm. For example, the execution model of languages such as Java and C++
promotes the synchronous communication model between objects. The paradigm of the lan-
guage(s) considered for implementing a system may influence the structure of the resulting
system at an architectural level (MEHTA et al., 2000a). 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” (Conway, 1968). 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 (Di Nitto and Rosenblum, 1999). In defining such a taxonomy, abstract platforms may need to be introduced in addition to physically existing platforms, which relieves 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 (Giesecke et al., 2007; Medvidovic, 2002; Medvidovic et al., 2003; Di Nitto and Rosenblum, 1999).

**Typical examples** Di Nitto and Rosenblum 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 (Luckham et al., 1995), Acme (Garlan et al., 2000) resp. Armani (Monroe, 2000), Wright (Allen and Garlan, 1997), ArchWare (Balasubramaniam et al., 2004; Oquendo et al., 2004), and Alfa (Mehta and Medvidovic, 2002, 2003). There has been important work on formalizing the properties of architectural styles using specifications in the general-purpose specification language Z (Abowd et al., 1995).

Garlan (1995) 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”.

6
3 Taxonomy

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. (1997), 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 (MONROE et al., 1997). 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 by ALEXANDER et al. (1977) as a vague metaphor to natural languages, but the same concept could better be referred to as a pattern vocabulary (cf. SCHMIDT et al., 1996).

Typical examples Abowd et al. (1995) discuss different variants of the Pipe-and-Filter and Event-based architectural styles that are tailored to specific needs.

Additional relevant literature KESHAV and GAMBLE (1998) 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 and KAZMAN (1999); KLEIN et al. (2000) 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. 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

\[\text{in the ISO 9126 terminology, it is probably more appropriate to speak of a quality (sub-)characteristic}\]
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 substiles 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 Clements et al. (2002), 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 refactoring, which are attached to refactoring 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.

**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. Grosskurth and Godfrey (2005). 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.
4 Characteristics of the Modes of Use

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 Bass et al. (1998)). 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. Garlan (1995)).

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 $y$. The modified architecture then conforms to both $x$ and $y$. This is denoted as $a \rightarrow x \rightarrow y$. A consistent style composition is one where for every $a \in A$ and $x, y, z \in S$, if $a \rightarrow x \rightarrow y$ and $a \rightarrow y \rightarrow z$, then $a \rightarrow x \rightarrow z$. This means that the composition is transitive. A style composition is consistent if and only if the composition of the style itself is consistent.
4 Characteristics of the Modes of Use

<table>
<thead>
<tr>
<th></th>
<th>Ad-hoc</th>
<th>Platform-oriented</th>
<th>Customized</th>
<th>Pre-modeling</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specialization</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Expliciation &amp; Rigor</td>
<td>Informal</td>
<td>No</td>
<td>Formal</td>
<td>Formal</td>
<td>Informal</td>
</tr>
<tr>
<td>Conceptual Level</td>
<td>(Abstract)</td>
<td>Technical</td>
<td>Abstract</td>
<td>Technical</td>
<td>Both</td>
</tr>
<tr>
<td>Relationship to System</td>
<td>No</td>
<td>Analyses</td>
<td>Analyses</td>
<td>Analyses</td>
<td>Analyses</td>
</tr>
<tr>
<td>Quality Characteristics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suitability for</td>
<td>Limited</td>
<td>Yes</td>
<td>Partially</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Architectural Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exploration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Characteristics of the Architectural Style Usages

$x$ and conforms to style $y$ as well. Then $x$ and $y$ are composable for $a$. Thus, there is a (partial) composition operator $\tilde{x}_1 : (A \times S) \times S \rightarrow A$ such that:

\[
(\tilde{x}_1(a, x, y) = b \Rightarrow b \text{ conforms to } y) \quad (1)
\]

\[
\land (\tilde{x}_1(a, x, y) \text{ defined } \Rightarrow a \text{ conforms to } x) \quad (2)
\]

Style composability at the type level is in place if a (partial) composition operator $\tilde{x}_2 : S \times S \rightarrow S$ 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 $a \tilde{x}_2 b$.

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 a 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 (GIESECKE et al., 2007).

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 (Giesecke, 2006b).

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 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 Di Nitto and Rosenblum (1999). 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 (Mehta et al., 2000b) or physical vs. logical architecture (Zendler and Schwartzel, 1998)).

Classification For ad-hoc usage, both types are possible. While the old general-purpose architectural styles (Garlan and Shaw, 1993) 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.
5 Conclusion

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 (Garlan, 1995). 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 (Giesecke and Bornhold, 2006) 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 (Giesecke, 2006b; Giesecke and Bornhold, 2006; Giesecke et al., 2007). 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.

References


Conway, M. E. (1968 April); How Do Committees Invent?; Datamation.
References


GIESECKE, S.; HASSELBRING, W.; and RIEBISCH, M. (2006); Classifying Architectural Constraints as a basis for Software Quality Assessment; Advanced Engineering Informatics. Special Issue on Ontology and Epistemology of Systems and Software Engineering. Conditionally accepted for publication.


References


KLEIN, M.; KAZMAN, R.; and NORD, R. (2000); A BASis (or ABASs) for Reasoning About Software Architectures, Software Engineering Institute.


References


VAN EMDE BOAS, P. (1997 February); Resistance is Futile; Formal linguistic observations on design patterns; Technical report ILLC-CT-97-02; ILLC, FWINS, Universiteit van Amsterdam; URL http://www.wins.uva.nl/research/illc/ResearchReports/CT97-02.text.ps.gz.
