Middleware-induced Architectural Style Modelling for Architecture Exploration

Simon Giesecke, Johannes Bornhold and Wilhelm Hasselbring
Carl von Ossietzky University of Oldenburg
Software Engineering Group
26111 Oldenburg, Germany
{giesecke,johannes.bornhold,hasselbring}@informatik.uni-oldenburg.de

Abstract

The MIDARCH Method is a software design method for exploring software architecture alternatives that use different middleware platforms. First, candidate architectures are modelled based on different architectural styles that are induced by the respective middleware platforms. Candidate styles are chosen using a taxonomy of such styles. The goal is the evaluation of candidate architectures with respect to a goal/question/metric quality model. We illustrate the modelling approach using the Apache Cocoon Web component framework and related technologies, and a taxonomy of their associated styles.

1. Introduction

The MIDARCH Method [5] aims to better exploit the benefits of architectural styles in migration, integration, or re-engineering projects for legacy middleware-intensive business information systems. It is a software design method for architecture-level design exploration.

A middleware technology induces one or more architectural styles [2]. Systems using a middleware technology are endorsed to adhere to these styles in order to benefit most from the technology. To avoid confusion with other uses of the term "architectural styles", we use the term *MINT Styles* (Middleware INTegration Styles) to refer to this type of architectural constraint. In practise, legacy systems often do not conform to a MINT Style strictly, among other reasons because the style is only implicitly described in the middleware documentation. To ensure the adherence to a MINT Style, it must (a) be explicitly described in a formal language in order to (b) allow checking the conformance of a concrete architectural description with the rules set up by the style.

Formalising the style improves understanding and communication about the global concepts underlying the architecture. Adherence to a coherent style improves understandability and maintainability of the system and its architecture. In the context of the MIDARCH Method, results of architecture evaluations can be related to the style used. Thus, the knowledge gained in evaluating architectures conforming to some style can be reused and exploited in further MIDARCH Instances.

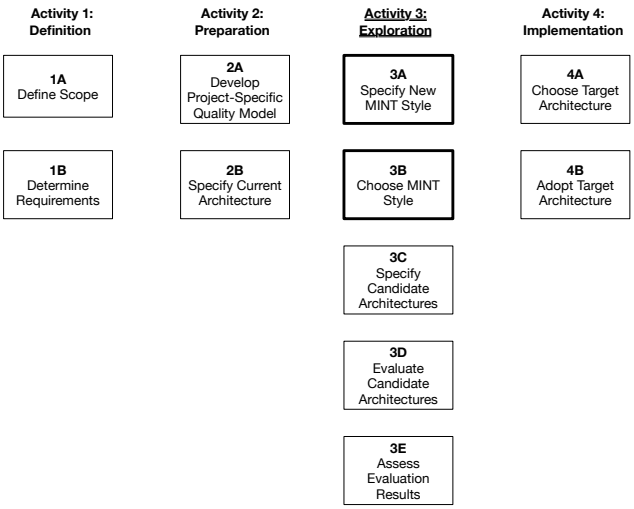

Figure 1. Activities of the MIDARCH Method

Outline In Section 2, we present the goals and activities of the overall MIDARCH Method. Then, the MINT Style modelling approach is described in detail in Section 3. Related work is discussed in Section 4 and the paper is concluded with an outlook on future work in Section 5.

2. Overview of the MIDARCH Method

In Figure 1, the four activities and their sub-activities of the MIDARCH Method are shown. In this overview, no dependencies between the sub-activities are displayed. The activities and sub-activities are not strictly sequential. Their

Proceedings of the Working IEEE/IFIP
Conference on Software Architecture (WICSA'07)
0-7695-2744-2/07 $20.00 © 2007 IEEE

exact dependencies are not shown here for brevity. The activities are described briefly in the following:

Definition The scope of the project, i.e. the systems involved, and the goals to be achieved are defined.

Preparation A project-specific quality model (in the sense of a goal/question/metric quality model) is developed, and the current architecture is modelled, if an architecture description does not exist in a suitable form.

Exploration This activity is central to the proposed method and is assumed to consume the majority of the effort required for an instance of the method. It involves the preselection of MINT Styles, modelling of candidate architectures that conform to the selected styles, and the evaluation of the resulting architectures. For evaluation, existing Architecture Evaluation Methods (such as ATAM [9]) are used. Finally, the evaluation results are assessed to decide whether an architecture candidate (or a set of such) that allows to achieve the stated goals has already been found.

Implementation Based on the previous activity, the architecture to be implemented is developed, which might involve combining multiple candidate architectures, and the implementation is performed.

The core contribution of the method, and its most distinctive feature, is the style modelling approach which is part of the Exploration activity. We focus on the MINT Style modelling sub-activity 3A in the next section.

3. MIDARCH Style Modelling Approach

An instance of the MIDARCH sub-activity 3A comprises modelling a new style based on a middleware technology and inserting it into a MINT Style Taxonomy.

We decided not to develop an entirely new ADL, but to evaluate the aptness of existing ADLs for our purposes and devise a method for modelling styles and style-based architectures using existing ADLs. In fact, we consider the aspect of the language used to represent the resulting styles as subordinate to the conceptual approach. We currently use Acme [4] for style modelling, but this can also be considered variable in each instance of the MIDARCH Method. However, we consider it essential to use a formal language which allows expressing both style definitions and conforming architectures.

We thus distinguish the (abstract) *concept* of MINT Styles from the (concrete) *notation* used for expressing MINT Styles.

Assumptions The architectural description of a software system usually consists of multiple models, which provide various views on a system from different viewpoints [8]. Each view may conform to a distinct architectural style [6].

We consider the application domain of web-based and distributed middleware-intensive business information systems. *Middleware-intensive* software systems are such systems, for which the choice of middleware technology significantly influences the structure of the resulting system. The architecture of such a system is usually considered to consist of three coarse-grained layers: Business Layer, Application Layer, and Infrastructure (or Technology) Layer (cf. [7]). For our purposes, we split up the Infrastructure Layer into at least three sublayers: Middleware, Operating System and Hardware Resources.

Challenges of Middleware-induced Style Modelling There are several challenges that are specific to modelling middleware-induced styles, compared to architectural style modelling in general. In this paper, we consider four of these challenges, which are introduced here, and our (preliminary) solutions are discussed in the following.

First, modelling the style induced by a middleware product should reflect all aspects that are visible at the architectural level from the logical component structure viewpoint. Services that are offered by the middleware should also be reflected in the style, while retaining the fact that they reside on a different architectural layer.

Second, styles induced by middleware technologies should be related to what is traditionally regarded as architectural styles, i.e. very generic architectural styles. These styles are well-known and knowledge on the relationship of middleware products to generic styles makes their understanding easier and more reliable. However, these represent quite vague structuring principles. It is thus difficult to formalise them such that any concrete MINT Style that seizes the idea underlying a generic style can be expressed as a strict specialisation of that style. Our MINT Style Taxonomy approach deals with this challenge.

Thus, the question of what combining architectural styles means becomes very prominent: The implementations of middleware technologies may be layered upon each other (e.g., Cocoon is based on Fortress). Should one style be modelled as a specialisation of the other or should they be specified as independent styles? This requires design choices when modelling individual styles.

The last challenge arises from the fact that we chose not to develop an entirely new ADL. This implies that compromises must be made in mapping the features we deem conceptually relevant to the concrete language elements of existing ADLs, since we subject us to the sets of language design choices that have been made by their designers. Thus, for each target ADL, a mapping of the MINT concepts to the ADL's concepts must be defined.

The MINT Style Concept The *MINT Style* concept is based on the ideas in [1, 3], where an architectural style is

Proceedings of the Working IEEE/IFIP
Conference on Software Architecture (WICSA'07)
0-7695-2744-2/07 $20.00 © 2007 IEEE

considered to consist of the following aspects:

1. A vocabulary of design elements, i.e. component and connector metatypes.
2. A set of configuration rules.
3. A semantic interpretation, which gives some well-defined meaning to all configurations of design elements that satisfy the configuration rules.
4. The definition of analyses for configurations of that style, e.g. schedulability and deadlock analysis.

The first two aspects can be codified in an ADL, the third is necessarily at least in part informal. At the moment, we do not focus on analyses. We use the term "metatype" instead of "type" at the style level, to allow the sole use of "type" for types defined at the instance level.

This style concept can be applied to any structure-intensive viewpoint. The viewpoint we use can be characterised as the *logical component structure* viewpoint on the Application Layer. The modelling elements and their relationships that can be used on this layer are provided by the adjacent underlying middleware layer. In addition, a middleware product typically also provides services via basic components to applications. In our view, these form also part of the MINT Style. This does not mean that the internal structure of the middleware product is modelled within the style, but only those aspects that are exposed to applications. Architectural descriptions based on a MINT Style are thus implicitly non-strictly layered.

General MINT Style Modelling Approach A MINT Style is modelled in five steps:

1. Informally describe all style-relevant aspects based on the available documentation of the middleware product and existing expert knowledge. This yields a list of natural language requirements for the style specification.
2. Identify possible specialisation relationships to generic styles.
3. Map the requirements to component and connector metatypes, composition rules and basic services elements.
4. Formulate a style from these elements in the ADL as an extension to the generic styles identified in step 2.
5. Check whether the resulting style is consistent. If necessary, relieve from strict specialisation relationships.

MINT Style Definition in Acme When selecting a notation for representing MINT Styles, it is clear that it will not be possible to express every aspect, that is informally connected with the idea behind the style, in a formal way. Probably, any choice of notation will be a trade-off between expressiveness and usability/analysability of style descriptions. As noted above, we currently use Acme.

Most of the conceptual modelling elements may be mapped to Acme in a straightforward way. A MINT Style is modelled as a *family*. Metatypes and types are both mapped to *(component, connector, role and port) type* declarations: all types declared within a family are considered metatypes, those declared in system declarations are considered types. This solution is not perfect, since it does not allow to declare non-metatypes for middleware layer components. Further architectural rules are modelled as *invariants*.

MINT Style Taxonomy A MINT Style Taxonomy relates multiple styles to each other, and thus enriches the information contained in the isolated style descriptions.

We represent MINT Style Taxonomies as UML Class diagrams with specific stereotypes. Essentially, there are two types of entities contained in the taxonomy, which represent *general-purpose*, *abstract* and *concrete* styles, respectively. Put simply, concrete styles are associated with a concrete middleware product, while general-purpose and abstract ones are not. Abstract styles are still bound to some specification, while general-purpose styles represent more fundamental structuring principles. General-purpose and abstract styles are denoted by corresponding stereotypes.

We currently distinguish three types of relationships:

strict inheritance This is the regular inheritance used to express specialisation between a generic and a concrete style or between concrete styles of different middleware technologies which build upon each other (e.g. Cocoon builds upon Fortress).

variant inheritance Variant inheritance is a specialisation of strict inheritance and is used to express the relationship between a specialised variant and a more basic style of a single middleware product.

builds upon When one middleware platform is implemented using another middleware platform, the styles endorsed by these platforms may be related as well. If the lower level style remains visible to the application developer, a *builds upon* relationship should indicate this fact.

Example Taxonomy As Figure 2 shows, we have identified the general-purpose style *Inversion of Control* as a superstyle for an abstract style *Avalon Framework* which mainly models the lifecycle concept for components. The *Inversion of Control* style will also be the superstyle for the MINT Style *Spring* induced by the Spring Framework, which has not yet been modelled. The MINT Style *Fortress* extends the *Avalon Framework* style by the definition of the specific container type, which is used to manage the components through their lifecycle. Both shown Cocoon styles build upon the *Fortress* style and loosely extend the generic *Pipe-and-Filter* style.

Proceedings of the Working IEEE/IFIP
Conference on Software Architecture (WICSA'07)
0-7695-2744-2/07 $20.00 © 2007 IEEE

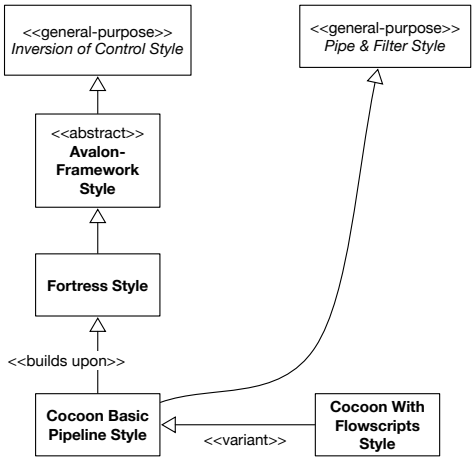

Figure 2. MINT Style Taxonomy of Cocoon-related Styles

4. Related Work

Related work most importantly includes other methods for modelling architectural styles (specifically, platform-induced styles) and style-based architectures.

The idea of explicitly linking middleware platforms with architectural styles in the sense that a middleware platform *induces* an architectural style was first described in [2]. We extend and refine their notion of middleware-induced styles. In [2], Armani/Acme and Rapide are evaluated for modelling the styles induced by JEDI and C2. We also use Armani/Acme, but consider a somewhat different kind of middleware technologies.

Focusing on middleware for distributed object architectures, [10] discuss the relationship of (implementation-level) middleware and software architecture. They argue that, when using the C2 style, variability in middleware can be best modelled by different connectors. However, they acknowledge that topological constraints may depend on the chosen middleware technology. In our work since we consider a different type of middleware technologies.

5. Conclusion and Future Work

In this paper, we briefly sketched the overall MIDARCH Method for supporting migration, integration, or re-engineering projects for legacy middleware-intensive business information systems. We described the general approach for modelling MINT Styles based on middleware technologies and creating a MINT Style Taxonomy. We showed a mapping to an existing Architectural Description Language (Acme), and discussed challenges and limitations of the current approach.

As future work, we have planned to model styles of further related middleware technologies (e.g. Spring and OSGi) and analyse their relationships for the extension in the MINT Style Taxonomy. Further real-life case studies evaluating the use of a MINT Style Taxonomy and the MIDARCH Method should be done. Several open questions of the style modelling approach must still be solved, which have been noted in Section 3.

For the purpose of selecting a style (sub-activity 3B), we plan to provide a MINT Style Repository tool which provides access to a MINT Style Taxonomy and the evaluation results of former MIDARCH Instances. With a rich MINT Style Taxonomy, it is possible to refine the chosen MINT Style in a stepwise approach.

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] E. Di Nitto and D. Rosenblum. Exploiting ADLs to specify architectural styles induced by middleware infrastructures. In *Proc. of the 21st Intl. Conf. on Software Engineering*, pages 13–22. IEEE Comp. Soc. Pr., 1999.

[3] D. Garlan. What is style? In *Software architectures*, volume 106 of *Dagstuhl-Seminar-Report*, Saarbrücken, Germany, February 1995.

[4] 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, 2000.

[5] S. Giesecke. A method for integrating enterprise information systems based on middleware styles. In *International Conference on Enterprise Information Systems (ICEIS'06) Doctoral Symposium*, pages 24–37. INSTICC Press, Portugal, 2006.

[6] S. Giesecke, W. Hasselbring, and J. Matevska. Extending the ANSI/IEEE Standard 1471 for the representation of architectural rationale. In *Nordic Workshop on the Unified Modeling Language (NWUML'06)*, 2006.

[7] W. Hasselbring. Information system integration. *Commun. ACM*, 43(6):32–38, 2000.

[8] ISO. *Recommended Practice for Architectural Description of Software-Intensive Systems*, 2006. IEEE Standard 1471-2000, ISO/IEC DIS 25961.

[9] R. Kazman, M. Klein, and P. Clements. ATAM: A method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University, 2000.

[10] 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.