Architectural Styles for Early Goal-driven Middleware Platform Selection

Simon Giesecke[1], Matthias Rohr[2] and Wilhelm Hasselbring[1,2]

[1] OFFIS Institute for Information Technology, Escherweg 2,
26121 Oldenburg (Oldb.), Germany,
simon.giesecke@offis.de
[2] Carl von Ossietzky University of Oldenburg, Software Engineering Group,
26111 Oldenburg (Oldb.), Germany,
{rohr,hasselbring}@informatik.uni-oldenburg.de

Abstract. The selection of a middleware platform is an important task in the development of complex business information systems. A systematic selection aims to ensure that the resulting system meets its specified quality requirements. The selection must be made early, since it significantly influences the software architecture and later changes to the architecture are expensive. In our approach, middleware platforms are explicitly described by architectural styles, which capture the design vocabulary and structural constraints the platform imposes upon concrete application architectures. This novel combination of architectural styles and middleware platform selection has the benefit that the selected style provides partial guidance for modelling the application architecture based on the chosen middleware platform. We present the MidArch method for systematic middleware platform selection exploiting a style repository which stores results of architectural evaluations from applications of the method. We present an example of applying the method in the development of a web-based information system, in the context of a field case study.

1 Introduction

The selection of suitable middleware products on top of which components are implemented or heterogeneous subsystems are integrated, is one critical task in the development process of current business applications. Besides functional considerations, an application architect needs to make a selection that enables the resulting system to meet given quality (or non-functional) requirements concerning characteristics such as reliability, usability, efficiency, maintainability or portability [1]. These requirements usually concern conflicting characteristics that require a trade-off. For example, different distribution middleware products impact the availability and the time efficiency of the application in different ways.

The selection task is critical because business information systems are middleware-intensive software systems, i.e., systems whose structure and behaviour are significantly influenced by the choice of middleware products [2]. Therefore, a

well-founded decision on using a middleware product should be made as early in the development process as possible, i.e. at the architectural level. However, few guidelines or specific techniques exist on how to select a middleware product in a given project context, which often leads to ad-hoc selection decisions [3, 4]. We consider a wide definition of middleware, which includes not only distribution middleware, but also application frameworks (cf. [5]).

The MidArch Design Method supports the systematic selection of a middleware platform best suited to attain given system quality requirements. A middleware platform is defined by a specific mode of employment of a middleware product. We describe middleware platforms formally by *Middleware-oriented Architectural Styles*, or MidArch Styles in brief, which capture the vocabulary of platform abstractions and structural constraints imposed upon application architectures based on the corresponding platform. They can be notated using the UML-based UML/MidArch approach.

When applying the MidArch Design Method, essentially two cases must be distinguished: Either an existing MidArch Style is chosen from a style repository, or, if no suitable style exists in the repository, a new MidArch Style is modelled. In previous papers, we concentrated on the specification of new styles and style-based architectures [6, 7]. There, MidArch Styles are used to guide modelling of a target application architecture: An application architecture based on the corresponding platform uses the design vocabulary of a MidArch Style and is required to formally conform to the rules of the style. In the course of one application of the MidArch Design Method, candidate architectures are modelled for several MidArch Styles and their quality is evaluated. The evaluation results for the candidate architectures are then lifted, such that they can be related to the MidArch Style and therefore to the middleware platform.

Initially, i.e. before the MidArch Method has been used in some project, the MidArch Repository is empty and contains no style definitions nor evaluation results. When the MidArch Method is applied at that point, no guidance is provided to the architect in preselecting a MidArch Style to model and apply in modelling an architecture. By acquiring more and more knowledge, guidance to the architect is steadily increasing. For this paper, we consider the case that all relevant MidArch Styles have already been specified and empirically evaluated to some extent through earlier instances of the MidArch Design Method. We demonstrate the reuse of the knowledge on the quality effects of styles in a new development project that uses the MidArch Design Method for selecting candidate MidArch Styles. The styles are chosen from a repository that stores MidArch Styles that are hierarchically organised in a taxonomy together with evaluation results. The focus and contribution of this paper are the method for this selection task and the underlying concepts.

Overview The remainder of the paper is structured as follows: First, we introduce foundations on software architecture and architectural styles in Section 2. Then, an overview of the overall MidArch Design Method is given in Section 3. The Style Selection Task of the MidArch Design Method is discussed in detail in

Section 4, which is the core contribution of this paper. In Section 5, related work is discussed, before Section 6 concludes the paper.

2 Software Architecture and Styles

Software architectures are abstractions of individual software systems, which can be organised in several layers (Section 2.1), whereas architectural styles characterise families of software architectures and systems (Section 2.2). MidArch Styles are a special form of architectural styles that describe middleware platforms (Section 2.3).

2.1 Software Architecture and Layers

In conformance with the ISO/IEC DIS 25961 Standard [**?**] for architectural description, we use the term "(software) architecture" to refer to the fundamental organisation of an individual software system:

> Software architecture is the "fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution." [?]

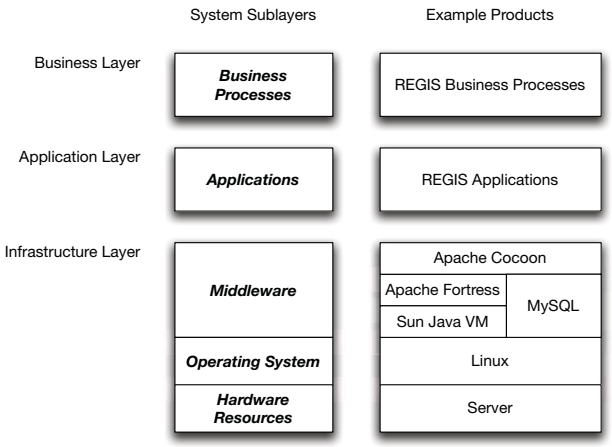

Fig. 1. System layers with example products for each layer

Business information systems are usually divided into three major system layers [8]: business process layer, application layer and infrastructure layer (see Figure 1). The infrastructure layer can be split into the basic hardware resources layer, a layer that consists of the hardware-specific operating systems, and a middleware layer, which may consist of diverse middleware platform products.

Schmidt et al. [5] consider four layers of middleware: domain-specific middleware services, common middleware services, distribution middleware (e.g. CORBA), and host infrastructure middleware such as a Java VM or a Common Language Interface VM. The latter imposes few structural constraints, but it is the main layer to attain the requirement of hardware portability.

We focus on *application architectures* on the application layer that build upon concepts defined by the underlying middleware layers, which should be reflected in the architectural description language used to describe middleware-intensive systems. Middleware-oriented architectural styles are a means to represent such middleware concepts. Before discussing specifics of middleware-oriented architectural styles, we first introduce the general concept of architectural styles.

2.2 Architectural Styles

An *architectural style* characterises a family of related software architectures [9]. We assume the component-and-connector viewpoint [10, chapter 3] of software architecture. From this viewpoint, an architectural style, such as the pipes-and-filters or layered style, characterises a family of configurations of components and connectors. A view from this viewpoint is principally described through such a configuration. An entire software architecture description typically comprises other views as well, which we do not focus on.

Common features of a family of related software architectures can be captured in architectural patterns or architectural styles[3]. These artefacts specify constraints for component and connector types, and rules for the composition of components and connectors into an architecture [12]. Architectural styles are typically specified in a formal Architecture Description Language (ADL) such as Acme [13], which allows the specification of both architectural styles and software architectures. The formal specification allows to analyse specifications of software architectures for their conformance to an architectural style.

An architectural style [14] consists of the definition of a vocabulary of component and connector metatypes, and of composition rules for configurations. Architectural styles can be used to guide the development of a software architecture, since they constrain the vast architectural design space through the provision of the vocabulary and its associated rules. They summarise a set of design decisions that are proven to fit together, and ease subsequent design decisions.

Conformance to an architectural style ensures internal conceptual coherence and consistency of an architecture. Moreover, specific quality properties of the resulting system may be achieved through using an architectural style, e.g. good scalability [15]. Thus, checking an architecture for conformance to some

[3] Architectural patterns and styles are similar concepts. We use the term "architectural style" as this is faithful to the tradition in the context of ADLs. Some authors, e.g. [11], refer to these common features as "architectures" as opposed to our use of that term, which may be confusing.

architectural style is an important architectural analysis[4]. To facilitate such a *style conformance check*, formalisms are required that allow the specification of both architectural styles and architectures. Several Architectural Description Languages (ADLs) that offer this feature have been developed, including Acme, Alfa, ArchWare and Wright.

Unfortunately, these formalisms are not well-known by practitioners, and even in the overall software engineering research community. Moreover, it is difficult to combine them with software modelling notations that are typically used to model software, most importantly the Unified Modeling Language (UML). Therefore, we have developed the UML/MidArch approach (see Section 2.3).

2.3 MidArch Styles

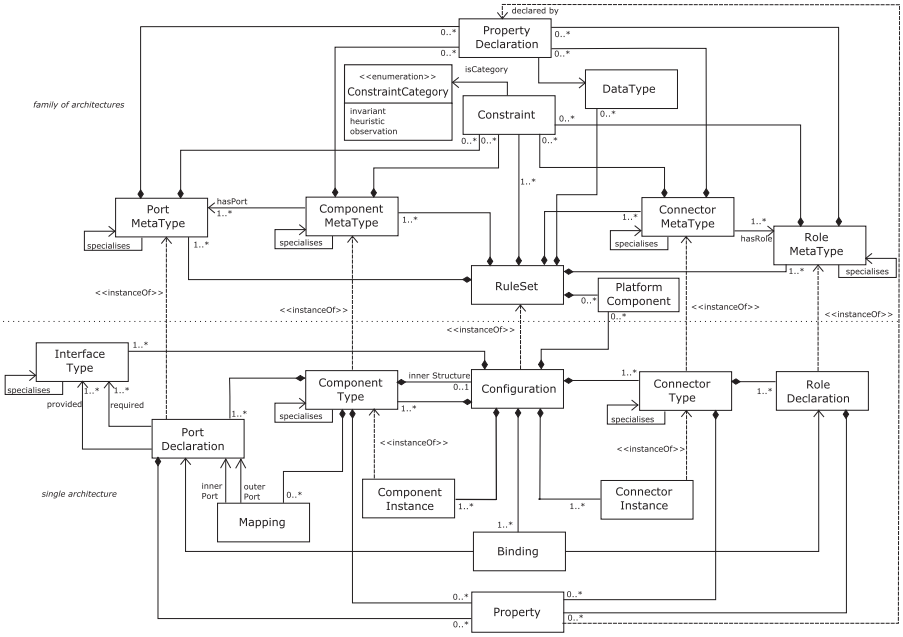

Fig. 2. Conceptual MidArch Style Metamodel

We give an overview of the MidArch Style Metamodel in Figure 2. It is partitioned by a dashed horizontal line, which separates the concepts concerning the family of architectures level (above the line) from the concepts concerning the

[4] Performing conformance checks is a distinguishing property of architectural styles in comparison with design patterns [16], which embody the same concepts but are defined archetypally only.

individual architecture level (below the line), from the component-and-connector viewpoint.

Architecture Family Level A MidArch Style, which is called RuleSet in the metamodel, is at the core of the style modelling level. Most importantly, it contains the following elements:

- ComponentMetaTypes, PortMetaTypes, ConnectorMetaTypes and RoleMetaTypes, which define the style vocabulary. PortMetaTypes are associated with ComponentMetaTypes, while RoleMetaTypes are associated with ConnectorMetaTypes. These only serve as a coarse classification of elements, and do not specify detailed interfaces, which are assumed to be application-specific.
- Constraints restrict the valid system level compositions of the elements based on that vocabulary.

Individual Architecture Level Application-specific types of components and connectors for a single architecture are specified and instantiated on the system level. Compositions of components, connectors, and their subordinate ports and roles can only be specified on the system level. Interfaces, i.e., lists of operation signatures, may be specified for ports, and components' ports are bound to connectors' roles.

UML/MidArch Notation While the metamodel above is at the conceptual level and does not prescribe a specific notation for architectural styles, we also developed a UML-based approach for modelling component-and-connector views, which has the additional benefit of easy integration with models for other architectural views, such as behavioural aspects, within a single notation. While the UML does not natively provide a modelling construct for architectural styles, we use UML Profiles to define an architectural style in the UML/MidArch modelling approach [**?**]. A UML Model that conforms to a style profile then describes an architecture.

Example: Apache Cocoon Framework Apache Cocoon is a "a web development framework built around the concepts of separation of concerns and component-based web development" [17]. Cocoon is designed as a Java Servlet. Requests are processed in a pipeline in which several components (*filters*) are hooked together, i.e. it uses a variant of the pipe-and-filter architectural style. Within the pipeline, filters communicate via a stream of SAX events. The entry to the pipelined processing is a *generator* followed by an arbitrary number of *transformers* and finalised by a *serialiser* which typically serialises the SAX events into an HTML output.

Figure 3 shows a fragment of the description of the basic Cocoon style in the UML/MidArch approach. Port and role metatypes as well as connection rules are omitted, only the defined component and connector metatypes are shown. It references two packages of the UML2 Superstructure [18] (Collaborations and

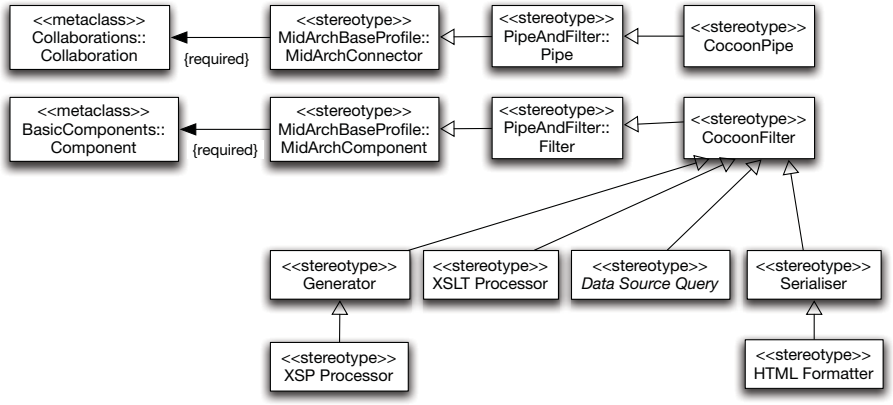

Fig. 3. Component and Connector Metatypes of the Cocoon Style in UML/MidArch

BasicComponents), the basic profile that underlies all style definitions in the UML/MidArch approach (MidArchBaseProfile) and a profile for the generic pipe-and-filter style (PipeAndFilter), which is extended by the Middleware-specific Cocoon style shown. Component metatypes are represented by stereotypes for the Component metaclass, while connector metatypes are represented by stereotypes for the Collaboration metaclass[5].

3 MidArch Design Method

In this section, we describe the overall MidArch Design Method, before we present its style selection task in detail (see Section 4). First, we discuss the rationale of the method (Section 3.1). Then we present the activities of the method (Section 3.2), and introduce the running example for this paper (Section 3.3).

3.1 Rationale

We already discussed in Section 1 that middleware selection is a critical task in the development of complex business information systems. Most current development projects do not start from scratch but are confronted with a set of existing applications. These applications must be modified on the basis of middleware products that are already used, integrated with newly developed components using some middleware product, or the existing applications should be migrated towards a new underlying middleware product. We assume that the middleware product itself already exists.

The selection of a middleware product must be complemented with the determination of a specific approach to employing the middleware product

[5] The reason for using Collaborations rather than Connectors are intricate [?].

consistently, as most middleware products allow different modes of employment (cf. [19]). These modes correspond to design alternatives that may influence the structure and behaviour of the application to the same extent as do different products. We refer to such a combination of a middleware product and its usage as a *middleware platform*. An application architect must choose between a set of such middleware platforms, not between products on the first hand. For example, Java Enterprise Edition applications can choose to use message-based communication (Message-driven Beans) rather than call-and-return communication (RPC). If different modes of employment must be combined, it is essential to use a well-defined combination to avoid architectural mismatches, which in turn ensures understandability and maintainability of the system.

The MidArch Approach consists of the MidArch Design Method, which uses MidArch Styles and a MidArch Taxonomy for the systematic selection of middleware platforms and for guiding the specification of software architectures.

MidArch Styles[6] capture the structural constraints imposed by a middleware platform. A MidArch Style characterises the family of software architectures that can be implemented on the associated middleware platform. It acts as a guidance for the process of modelling a software architecture for a specific system.

We consider MidArch Styles as a language-independent concept, but define mappings to the ADL Acme [6] and to UML Component Diagrams [?]. All known MidArch Styles are organised within a MidArch Taxonomy, which identifies specialisation and other relationships among the MidArch Styles.

Within the MidArch Design Method, a MidArch Taxonomy provides the basis for the stepwise selection of a suitable middleware platform. An architect specifies software architectures according to a MidArch Style and evaluates them against given quality requirements. The evaluation results are stored as annotations to the related MidArch Style in the MidArch Taxonomy. These annotations can be used in later middleware platform selections to improve the selection process. The knowledge base on MidArch Styles continuously accumulates knowledge with each application.

For modern complex middleware products, it is non-trivial to extract the architectural rules that are necessary for an effective use of the middleware product, as various software development projects have shown [3, 7, 22]. Even when these rules are known, they must still be applied consistently in the design of the architecture and in the actual implementation. The MidArch Approach ensures the consistent application of these rules within the architecture, and also supports their enforcement in the implementation.

3.2 Activities

In Figure 4, the four *activities* of the overall MidArch Design Method and their constituent *tasks* are shown. The activities and tasks are not strictly sequential. In the *Definition* activity, the scope of the project, i.e. the systems involved,

[6] MidArch Styles follow the line of research on architectural styles as represented by [9, 14, 20, 21].

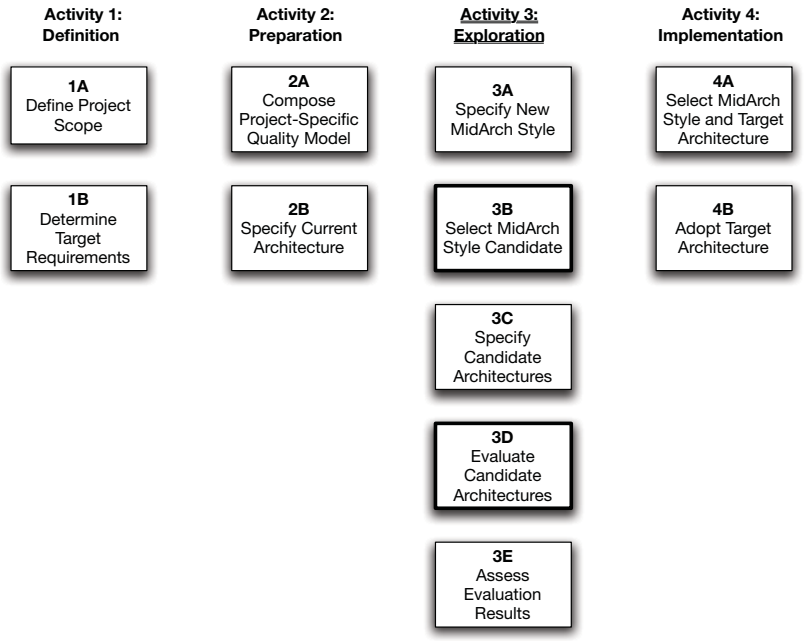

Fig. 4. The Activities and Tasks of the overall MidArch Design Method

and the goals to be achieved are defined. In the *Preparation* activity, a project-specific quality model (in the sense of a goal/question/metric quality model [23]) is developed on the basis of partial quality model profiles, and the current application architecture is modelled if a legacy system already exists, but an architecture description is not available in a suitable form. Next is the *Exploration* activity, which is central to the proposed method and is assumed to consume the majority of the effort required. It involves the preselection of MidArch 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 [24]) 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 been found. On this basis, the architecture to be implemented is developed in the *Implementation* activity, which might involve combining multiple candidate architectures, which is then implemented.

We illustrate the tasks of the MidArch Design Method using an example application of the method. This also serves as a running example for Section 4, where Tasks 3B and 3D (highlighted in Figure 4) are discussed in more detail.

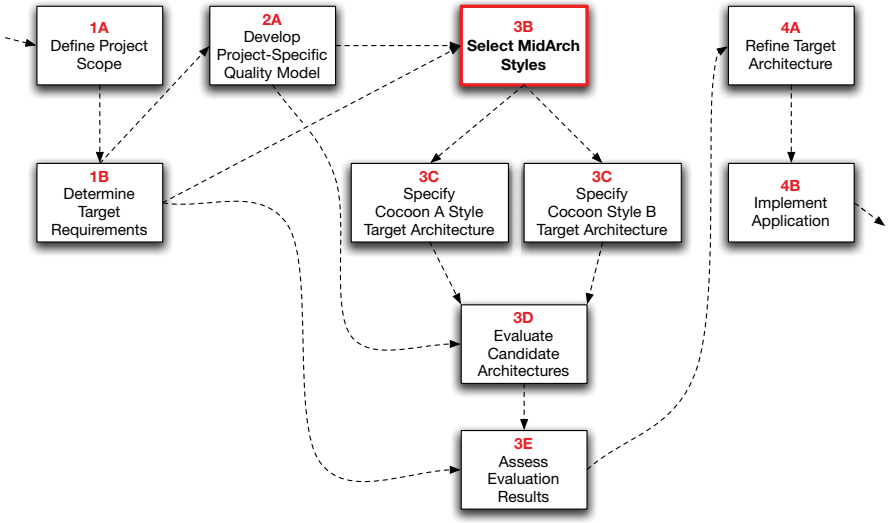

Fig. 5. An Example Application of the MidArch Design Method

3.3 Running Example

Figure 5 shows a simple example application of the MidArch Design Method, which also depicts the dependencies between the tasks. The dependency arrows have a completion (finish-to-finish) semantics, i.e., the target task cannot be completed before the source task is completed. However, for simplicity, we discuss the process as if the tasks were performed sequentially.

First, the project scope is defined in Task 1A, which is the development of a new web-based information system. In Task 1B concrete target requirements are identified. The goals covered by these target requirements are refined into a detailed quality model in Task 2A. An excerpt of the simple quality model for this example is later shown in the top part of Figure 10. In a real application of the MidArch Design Method, a full-fledged goal/question/metric quality model should be used.

In general, a quality model in the MidArch Design Method may exhibit a wide range of metrics, ranging from expert judgements to formal prediction techniques. The degree of detail of the architecture models significantly impacts the accuracy of the results but also the cost of modelling and evaluation. Our research does not provide an original contribution to architecture evaluation techniques. Therefore, we in order to keep the evaluation simple in this paper, we use the following example: the quality characteristics maintainability, scalability and availability are each judged by an expert on a simple ordinal 5-level scale. The experts are free in their judgment, so the result is obviously subjective, while can be made more reproducible by using some guideline for the judgment. In

addition, intersubjectivity can be achieved by including multiple experts in the judgment process.

In Task 3B, the target requirements—stated in terms of the quality model—are matched against the evaluation results for the styles contained in the MidArch Repository. We assume that several styles have already been added to the MidArch Repository and evaluation results for these styles have been obtained in previous MidArch instances.

The matching procedure yields two candidate styles termed Cocoon A and Cocoon B. Both of these styles are based on the Cocoon Basic Pipeline Style, but they differ in the realisation of database accesses: Cocoon A requires all database accesses to be encapsulated within one filter of the pipeline, while Cocoon B does not. More details on this task are discussed in Section 4.4. For both of these styles, candidate architectures are modelled in the two occurrences of Task 3C.

Task 3D then applies an architecture evaluation method to both candidate architectures. The results are compared and added to the MidArch Repository, as explained in Section 4.3. In Task 3E, an overall assessment of the evaluation results is done, which evaluates the results against the requirements and ranks the candidate architectures.

In Task 4A, the final target architecture is determined and a mapping to the implementation level artefacts is defined. The implementation is finally carried out in Task 4B.

4 Evaluation and Selection of Middleware-oriented Architectural Style

In this section, we elaborate the style selection task (Task 3B in Figure 4) and its prerequisite, the evaluation of style-based architectures (Task 3D). In the MidArch approach, the evaluation of MidArch Styles is performed *indirectly* and *comparatively.*

First, the evaluation is indirect, i.e. the evaluation is not applied directly to the style descriptions, but to artefacts that conform to the styles. An indirect evaluation can either use scenarios that are considered typical for that style, or architectures from concrete real projects. In the MidArch Design Method, we chose the second possibility. In our opinion, a direct evaluation of styles is not feasible, i.e., applying some evaluation technique to style descriptions only, i.e., without considering other input, would not return meaningful results.

Second, the evaluation is comparative, i.e.:

1. Multiple (at least) two candidate architectures for the same system specification need to be modelled, which conforming to different MidArch Styles each (Task 3C),
2. These candidate architectures are then evaluated (Task 3D), which yields evaluation results for each architecture. The metrics defined by the quality model are thus only applied to a single architecture description at once.

3. Now, the evaluation results are compared to determine the commonalities and significant differences between them. These differences are then associated with the MidArch Styles the candidate architectures conform to.

and the significant differences between the evaluation results are determined through comparison.

Section 4.1 discusses the role of the style selection task in the MidArch Design Method in detail on the levels of tasks. The relationship of MidArch Styles to Middleware products and platforms is analysed in Section 4.2. Section 4.3 explains how the evaluation results are obtained and annotated to style within a MidArch Taxonomy. Finally, section 4.4 discusses the use of these evaluation results in selecting a style from a MidArch Taxonomy.

4.1 Role within the MidArch Design Method

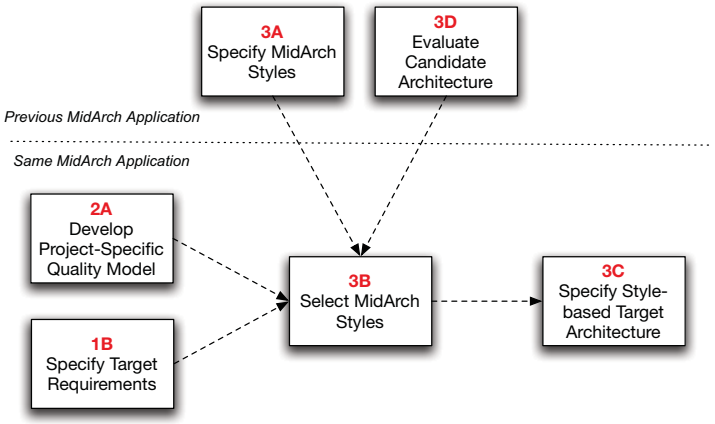

Fig. 6. Dependencies of Style Selection Task

Figure 6 illustrates the dependencies of the style selection task (Task 3B) to other tasks of the MidArch Design Method. These dependencies must be distinguished into those within the same MidArch application, and those across MidArch applications. Indirect dependencies that exist through other tasks are not shown here.

Dependencies and reuse across MidArch applications To consider a style for selection, it must have been specified first, and its effects on the quality of conforming architectures must be known to some extent. The style selection task depends on both the style specification task (Task 3A) and the architecture evaluation task (Task 3D). Task 3D is typically performed many times for a style,

however it must occur at least once per style. Task 3A, on the other hand, can be considered to occur exactly once for each style. If a style needs to be revised later, the evaluation results that have been acquired so far could be invalidated. Therefore, task 3A is critical and should be performed with great care.

Dependencies within the same MidArch application Within the same MidArch application, there exists a dependency on the specification of the target requirements (Task 1B) and development of the quality model (Task 2A). The target requirements need to be stated in terms of that quality model to serve as an input for matching against the comparative evaluation results.

A dependency in the other direction, i.e. a dependency on Task 3B, is that the specification of a candidate architecture based on a certain style (Task 3C) depends on the selection of that style.

4.2 Relationship of MidArch Styles, Middleware Platforms, and Middleware Products

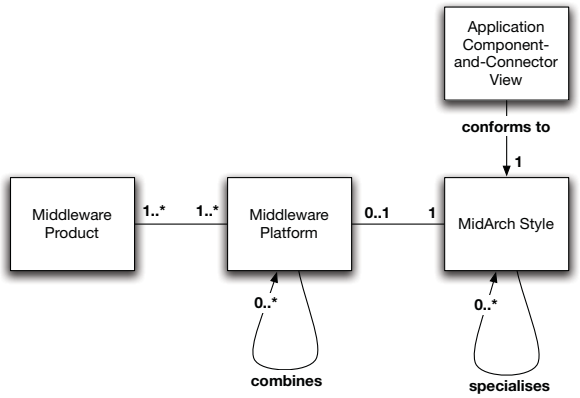

Fig. 7. Relationship of MidArch Styles and Middleware Platforms and Products

Figure 7 shows the relationships among middleware products, middleware platforms and MidArch Styles at class level: The relationship of middleware products and middleware platforms is not restricted. There can be many middleware products that are considered equivalent, e.g., different Java Enterprise Edition Application Servers[7]. On the other hand, there can be many modes of employment for a single middleware product. In addition, a middleware platform may be a combination of multiple constituent platforms that correspond to a

[7] Of course, different application servers may show subtle, but important functional differences, which may be considered important by a software architect. However, for simplicity we assume here that this is not the case.

combination of different products, e.g., a Java Enterprise Edition application using Hibernate.

While the relationship of middleware products and platforms is very flexible, we assume a one-to-one relationship of middleware platforms and (concrete) MidArch Styles, i.e., a (concrete) MidArch Style describes exactly one middleware platform. A MidArch Style may specialise other styles, but this relationships is much more restricted than the combination of middleware products, as a style is a formal artefact, for which a well-defined relationship is defined. Also, an application architecture's component-and-connector view is specified to conform to exactly one MidArch Style[8]. Thereby we relieve our theory from the need to provide a calculus of style combinations at the application architecture level, which we do not deem feasible.

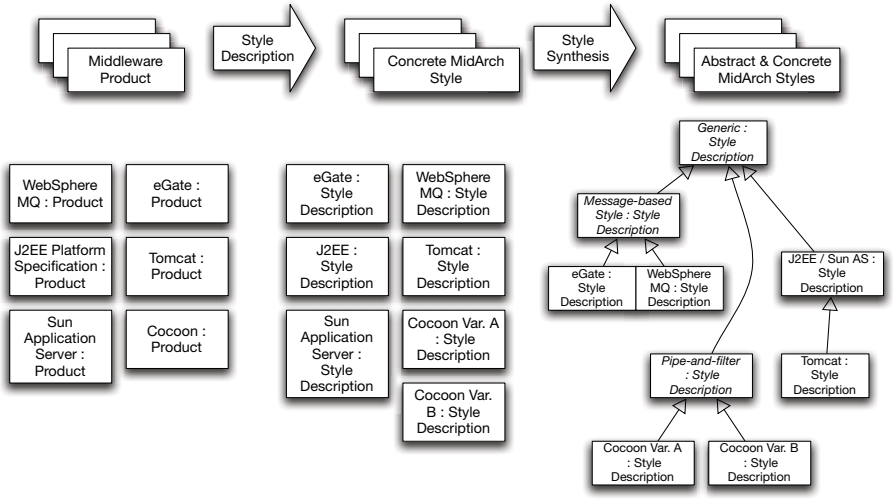

Fig. 8. Relationship of MidArch Styles and Middleware Platforms and Products: Example

Figure 8 shows example middleware products and the MidArch Styles modelled for them. We deliberately left out the corresponding platforms from the figure, because of the one-to-one relationship between platforms and styles. The rightmost area shows an organisation of the styles into a taxonomy. For the purpose of defining the taxonomy, additional styles have been introduced, which are considered abstract since they do not correspond immediately to a platform.

[8] Component-and-connector views that do not conform to any style at all are out of the scope of our current discussion. A view may accidentally conform to other styles that are not explicitly declared, but we do not consider such implicit relationships further either.

4.3 Comparative Evaluation of MidArch Styles

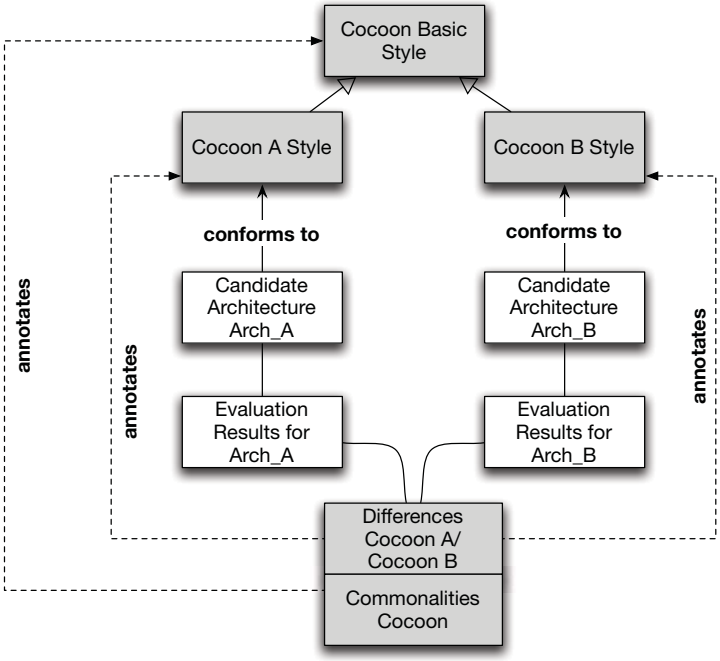

Fig. 9. Comparative Evaluation Process of MidArch Styles

Figure 9 gives an overview of the pairwise comparative evaluation process of MidArch Styles. The figure clearly separates the entities that are on the style level, which are shown with a dark background, and those on the architecture level, which are shown with a white background.

The three elements at the top represent a fragment of a MidArch Taxonomy, which contains the three MidArch Styles discussed in Section 3.3.

In the row below, the two candidate architectures resulting from the two occurrences of Task 3C (cf. Section 3.3) are shown, which *conform to* the respective MidArch Style. These architectures are evaluated, which yields the evaluation results in the row below the architectures. The evaluation results are still at the architecture level.

The evaluation results are raised to the style level by comparison. They are separated into a part containing measurements that are significantly different between the architecture evaluations *(differences)*, and a part containing the identical or very similar measurements *(commonalities)*. These parts need not cover all evaluation results, i.e., metrics that are neither significantly different nor similar for all evaluated architecture may be dropped.

The differences and the commonalities for the evaluation results for the two candidate architecture for the current system are shown at the bottom of Figure 9.

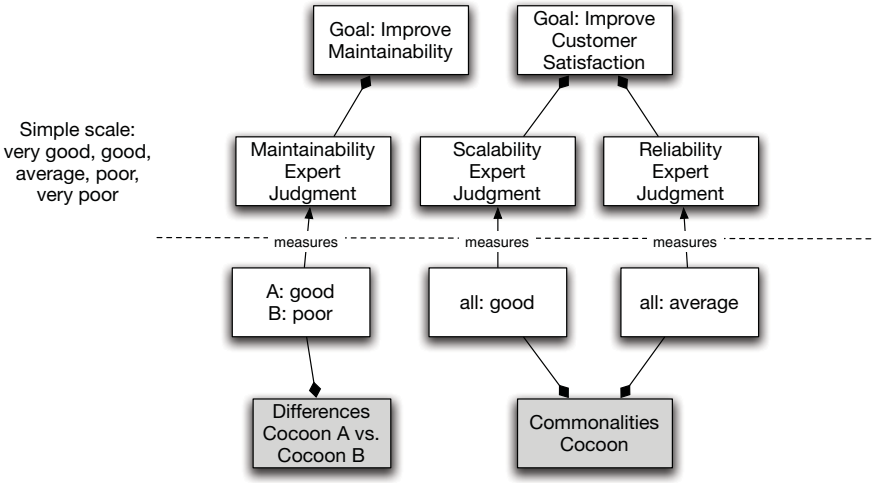

Fig. 10. Comparative Result of the Evaluation of Two MidArch Styles

Figure 10 illustrates their contents. In the top of the figure, above the dashed line, the metrics level of the quality model resulting from Task 2A is shown. Below the dashed line, the measurements of the architectures for the given metrics are shown and associated to either the differences or the commonalities.

4.4 Taxonomy-based MidArch Style Selection

Based on the evaluation described in Section 4.3, the selection of a suitable style from a MidArch Taxonomy can be performed. Figure 11 shows a MidArch Taxonomy, which contains the styles (depicted as rectangles) that will finally be selected as discussed above in Section 3.3 and in addition the abstract MidArch Styles Blackboard and Pipe & Filter. Evaluation results that are annotated to the styles are shown (depicted as ellipses). These are distinguished into those obtained directly by architectural evaluations as described in Section 4.3, and those that are derived indirectly from differences and commonalities of other evaluations.

The selection process then matches the requirements (depicted as rounded rectangles) stated in terms of the quality model against these evaluation results. The matching starts at the top of the MidArch Taxonomy and proceeds along the specialisations down to its leaves.

The algorithm shown in Figure 12 describes how the selection process works. The procedure “form clusters” can use different clustering algorithms, for example

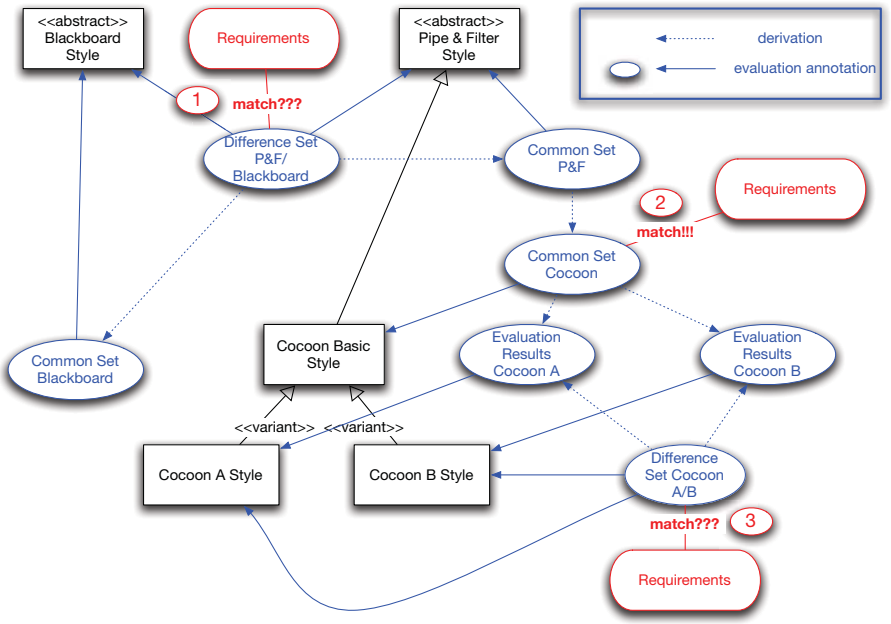

Fig. 11. Incremental Style Selection from a MidArch Taxonomy

SelectStyle(MidArchTaxonomy t, Requirements r) **returns** set of MidArchStyle
$current \leftarrow$ {root node of t}
while $\exists s \in current$: substyles(s in t) $\neq \emptyset$ **do**
  $next \leftarrow \emptyset$
  for all $s \in current$ **do**
    $current \leftarrow$ substyles(s in t)
    form clusters of $current$ by matchCommonalities(q, r, s, t)
    order $current$ first by the cluster and second within each cluster
      using the partial order induced by matchDifferences($q, r, s1, s2, t$)
    $next \leftarrow next \cup$ (first n styles from $current$)
  end for
  {perhaps sort out styles below a certain commonality match threshold value}
end while
{now all styles in $current$ are leaf nodes in the taxonomy t}
{propose $current$ as candidates for modelling architectural candidates}
return $current$

Fig. 12. Algorithm for Selecting a Style from a MidArch Repository

forming buckets of a fixed number of elements, or buckets with fixed match value interval.

Example In the example, the selection process proceeds as follows (the step numbers refer to the small ellipses in the figure):

1. The requirements are matched against the difference set of the abstract pipe-and-filter style and the abstract blackboard style. The result is that the pipe-and-filter style better achieves the requirements.
2. The only substyle of the pipe-and-filter style is the Cocoon Basic Style, which matches the requirements. Since this is already a concrete style, the matching could stop here, however we assume that the variants are also considered.
3. The matching process thus continues with the difference set of the Cocoon A & B variants. However, the result does not clearly suggest one or the other of the variants, so both are used to construct candidate architectures for subsequent evaluations in Task 3D (see Figurey 5).

5 Related Work

The methods we presented can be understood from two viewpoints, depending on whether one focuses on the first phase of acquiring the data, i.e. the evaluation of styles, or the second phase of using this data for making a decision on the selection of a middleware platform. The first phase constitutes an architectural evaluation method, and can thus be compared to other such methods. The second phase is a special case of the general problem of technology or COTS component acquisition. We discuss related work in these two areas in Sections 5.1 and 5.2. The distinction between the two aspects is not sharp, since any architecture evaluation method is aims making a decision, and any technology selection procedure should consider architectural issues. Finally, we discuss topics that are adjacent to our research, but not directly comparable in Section 5.3.

5.1 Architecture Evaluation

Besides the various general architecture evaluation methods, such as ATAM [24], ALMA [25], or several others [26], the middleware-specific architecture evaluation method MEMS has been developed by Liu et al. [19]. MEMS does not address exactly the same goal as MidArch, as the evaluation is not bound to specific projects, but to architectural scenarios that are considered typical for using the middleware product. The scope of MEMS ends with the creation of evaluation results. The valuation of these results in making a decision is expressly not covered.

Liu et al. argue that the general architecture evaluation methods do not readily address the problems raised by middleware. MEMS can essentially be understood as a middleware-specific parametrisation of some phases of the ATAM method. MidArch, however, is more Middleware-specific than MEMS in that it

involves explicit models of the structural aspects imposed by the Middleware platform as formal artefacts, i.e. MidArch Styles.

Zarras [27] discusses an application-independent comparison framework for middleware platforms, which also uses the notion of the architectural style imposed by the platform. However, they also subsume features that are not formally modelled under the notion of style.

5.2 Middleware Selection

Middleware selection is a special case of COTS component or technology selection/acquisition. Techniques for supporting this process can be distinguished into more business-oriented methods that focus on costs and benefits, and more technical methods that focus on quality characteristics. Business-oriented methods often view the selection as a decision in terms of decision analysis, e.g., [28]. Strategic and enterprise-wide considerations are only approached by business-oriented methods, while structural and behavioural details of the considered technologies are only considered by the technically-oriented methods.

Liu et al. [29] propose the i-Mate method for selecting middleware, which is similar to our approach, in that it involves the identification of candidate platforms from a knowledge base, and matches them against application-specific weighted requirements. In addition, they consider generic, application-independent requirements. The evaluation process consists of multiple steps of increasing depth, starting with coarse-grained scenarios for a greater number of candidates, towards prototyping for a shortlist of candidates. Methods similar to i-Mate include [30] and [31].

Goedicke and Zdun [32] perform a feature-based evaluation of several middleware platforms, which is aimed at the enterprise level rather than a single project level, and thus does not consider application-specific concerns.

5.3 Adjacent Topics

We are concerned with the choice among multiple middleware products that differ not only in their implementation, but in the structural constraints they impose on applications. If two middleware products impose the same constraints upon applications, they support the same middleware platforms and are thus considered equivalent in our approach, e.g., multiple Java Enterprise Edition Application Server products. The task of making a choice between multiple such products is addressed by middleware benchmarks (e.g., [33]).

Classifications concerning specific functional features or non-functional characteristics are not considered here (e.g., [34]). These results may be used as additional sources for the knowledge stored in the MidArch Repository (similar to [29]). However, in this paper, we are concerned with the underlying method that is not specific to any particular architectural characteristic.

6 Conclusions

In the paper, we have discussed some aspects of the MidArch Design Method for selecting middleware platforms based on middleware-oriented architectural styles (MidArch Styles). We focused on two tasks of the MidArch Design Method:

1. The comparative evaluation of MidArch Styles which is performed indirectly through the evaluation of style-based architectures, and
2. the selection of a suitable middleware platform through a stepwise selection process from a MidArch Repository, which stores evaluation results and MidArch Styles that are organised hierarchically in a taxonomy.

We demonstrated the applicability of these tasks using an example which builds upon an earlier industrial case study [7].

The current practise of middleware selection rarely uses systematic decision procedures: Existing methods for architecture evaluation are not employed, probably because they do not adapt well to the specifics of middleware. The MidArch Design Method specifically addresses middleware-specific concerns, since it is based on explicitly modelling the design vocabulary and structural constraints imposed upon applications by architectural styles. It thus offers the potential of improving the state of the practise. The MidArch Design Method has already been partially validated in a case study using an industrial application [6, 7]. A further contribution is the novel combination of the fields of architectural styles and middleware platform selection, which has the benefit that the selected style provides partial guidance for modelling the application architecture on the chosen middleware platform.

The evaluation results on the quality properties of middleware platforms obtained by the MidArch Design Method could be used to improve the design of future middleware technologies that adapt well to the requirements of actual applications.

The approach of lifting evaluation results for individual architectures to the style level builds upon the assumption that the influence of the architectural style on the quality of architectures is not overshadowed by variations between different system purposes. This will only be valid for systems that are sufficiently similar, e.g. regarding the application domains. Classifying the evaluation results for such classes of systems could improve the accuracy of the selection. However, the identification of system classes is an unsolved challenge of software engineering in general.

References

1. ISO/IEC JTC 1/SC 7: ISO/IEC 9126-1: Software Engineering – Product Quality – Part 1: Quality Model. (June 2001) Published standard.
2. Sutton, Jr., S.M.: Middleware selection. [35] 2–7
3. Ploski, J., Giesecke, S.: When small outgrows beautiful – experiences from a development project. In: Net.ObjectDays 2005, tranSIT (2005) 367–380
4. Di Nitto, E., Rosenblum, D.S.: On the role of style in selecting middleware and underwear. In Emmerich, W., Gruhn, V., eds.: Proceedings of ICSE Workshop on Engineering Distributed Objects (EDO99). (1999) 78–83

5. Schmidt, D.C., Buschmann, F.: Patterns, frameworks, and middleware: their synergistic relationships. In: ICSE '03: Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society (2003) 694–704
6. Giesecke, S., Bornhold, J., Hasselbring, W.: 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)
7. Giesecke, S., Bornhold, J.: Style-based architectural analysis for migrating a web-based regional trade information system. In Trentini, A., Marchetto, A., Bellettini, C., eds.: First International Workshop on Web Maintenance and Reengineering (WMR 2006). Volume 193 of CEUR Workshop Proceedings. (2006) 15–23
8. Hasselbring, W.: Information system integration. Commun. ACM **43**(6) (2000) 32–38
9. Garlan, D.: What is style? In Garlan, D., ed.: Software architectures. Volume 106 of Dagstuhl-Seminar-Report., Saarbrücken, Germany (February 1995) Proceedings of the Dagstuhl Workshop on Software Architecture.
10. Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., Little, R.: Documenting Software Architectures: Views and Beyond. Pearson Education (2002)
11. Jones, C.: Economics of software reuse. Computer **27**(7) (1994) 106–107
12. Shaw, M., Garlan, D.: Software architecture: perspectives on an emerging discipline. Prentice-Hall, Inc. (1996)
13. Garlan, D., Monroe, R.T., Wile, D.: Acme: architectural description of component-based systems. In: Foundations of component-based systems. Cambridge University Press, New York, NY, USA (2000) 47–67
14. Monroe, R.T., Kompanek, A., Melton, R., Garlan, D.: Architectural styles, design patterns, and objects. IEEE Software **14**(1) (January 1997) 43–52
15. Klein, M., Kazman, R.: Attribute-based architectural styles. Technical Report CMU/SEI-99-TR-022, Software Engineering Institute, Carnegie Mellon University (1999)
16. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc. (1995)
17. Apache Foundation: Apache Cocoon. `http://cocoon.apache.org/` (2006)
18. Object Management Group: UML 2.0 Superstructure Specification. (2005) OMG Specification formal/05-07-04.
19. Liu, Y., Gorton, I., Bass, L., Hoan, C., Abanmi, S.: Mems: A method for evaluating middleware architectures. In: Second International Conference on the Quality of Software Architectures (QoSA 2006). Volume 4214 of Lecture Notes in Computer Science. (2006) 9–26
20. Kim, J.S., Garlan, D.: Analyzing architectural styles with Alloy. In: ROSATEA '06: Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis, New York, NY, USA, ACM Press (2006) 70–80
21. Clements, P., Garlan, D., Little, R., Nord, R., Stafford, J.: Documenting software architectures: views and beyond. In: Proceedings of the 25th international conference on Software engineering, IEEE Computer Society (2003) 740–741
22. Garlan, D., Allen, R., Ockerbloom, J.: Architectural mismatch or why it's hard to build systems out of existing parts. In: Proceedings of the 17th international conference on Software engineering, ACM Press (1995) 179–185

23. Basili, V., Caldiera, G., Rombach, D.: Goal question metric paradigm. In Marciniak, J.J., ed.: Encyclopedia of Software Engineering, Volume I. John Wiley & Sons (1994) 528–532
24. Kazman, R., Klein, M., Clements, P.: ATAM: A method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University (2000)
25. Bengtsson, P., Lassing, N.H., Bosch, J., van Vliet, H.: Architecture-level modifiability analysis (ALMA). Journal of Systems and Software **69**(1-2) (2004) 129–147
26. Dobrica, L., Niemelä, E.: A survey on software architecture analysis methods. IEEE Trans. Softw. Eng. **28**(7) (2002) 638–653
27. Zarras, A.: A comparison framework for middleware infrastructures. Journal of Object Technology **3**(5) (2004) 103–123
28. Kazman, R., Asundi, J., Klein, M.: Quantifying the costs and benefits of architectural decisions. In: ICSE '01: Proceedings of the 23rd International Conference on Software Engineering, IEEE Computer Society (2001) 297–306
29. Liu, A., Gorton, I.: Accelerating COTS middleware acquisition: The i-Mate process. IEEE Softw. **20**(2) (2003) 72–79
30. Comella-Dorda, S., Dean, J.C., Morris, E.J., Oberndorf, P.A.: A process for COTS software product evaluation. In Dean, J.C., Gravel, A., eds.: ICCBSS 2002. Volume 2255 of Lecture Notes in Computer Science., Springer (2002) 86–96
31. Kontio, J.: A case study in applying a systematic method for COTS selection. In: ICSE '96: Proceedings of the 18th international conference on Software engineering, IEEE Computer Society (1996) 201–209
32. Goedicke, M., Zdun, U.: A key technology evaluation case study: Applying a new middleware architecture on the enterprise scale. [35] 8–26
33. Ran, S., Palmer, D., Brebner, P., Chen, S., Gorton, I., Gosper, J., Hu, L., Liu, A., Tran, P.: J2EE technology performance evaluation methodology. In: Distributed Objects and Applications 2002 (DOA'02), Proceedings (addendum). (2002) 13–16
34. Maheshwari, P., Pang, M.: Benchmarking message-oriented middleware: TIB/RV versus SonicMQ. Concurr. Comput. : Pract. Exper. **17**(12) (2005) 1507–1526
35. Emmerich, W., Tai, S., eds.: Second International Workshop on Engineering Distributed Objects (EDO 2000), Revised Papers. Volume 1999 of Lecture Notes in Computer Science. Springer (2001)