A STYLE-BASED ARCHITECTURE MODELLING APPROACH FOR UML 2 COMPONENT DIAGRAMS

Simon Giesecke
OFFIS Institute of Information Technology
Escherweg 2
26121 Oldenburg (Oldb.), Germany,
simon.giesecke@offis.de

Florian Marwede
Carl von Ossietzky University of Oldenburg
Software Engineering Group,
26111 Oldenburg (Oldb.), Germany,
florian.marwede@informatik.uni-oldenburg.de

Matthias Rohr
Carl von Ossietzky University of Oldenburg
Software Engineering Group,
26111 Oldenburg (Oldb.), Germany,
rohr@informatik.uni-oldenburg.de

Wilhelm Hasselbring
Carl von Ossietzky University of Oldenburg
Software Engineering Group,
26111 Oldenburg (Oldb.), Germany,
hasselbring@informatik.uni-oldenburg.de

ABSTRACT
There has been a variety of work arguing that the UML 1.x was not adequate as an Architectural Description Language (ADL). UML 2.x has made a large step towards supporting modelling software architectures, e.g. through the introduction of modelling constructs for composite structures. But many modelling constructs from ADLs can still not be mapped to the UML in a straightforward way. Important examples are typed connectors and architectural styles. We propose a mapping of of the MidArch modelling approach, which is similar to the well-known ADL Acme, to the UML 2.x and the OCL. The mapping is based on mapping the style/family construct to UML Profiles and the system/configuration construct to UML Component Diagrams. Architectural constraints are directly mapped to OCL constraints. In addition, OCL constraints are used to specialize the architecture modelling constructs of UML 2.x, and thus adapt the UML 2.x to the MidArch modelling approach. We describe a case study applying our approach and discuss our experiences.

KEY WORDS
Software Architecture, Architecture Description Language, Architectural Style, UML Profile.

1. Introduction

In version 2.0 of the Unified Modeling Language (UML), the OMG introduced new diagram types and a refined metamodel aimed at modelling software architectures, in particular the component-and-connector architectural view. This step has been made as part of a general movement towards broader applicability of the UML and as a response to criticism that the component modelling capabilities of the UML 1.x were not suitable for modelling software architectures (see, e.g., [9,10]). However, certain architecture modelling constructs such as typed connectors and architectural styles are still not part of the UML 2.0.

Over the 1990s, Architectural Description Languages (ADLs)[1] have been proposed for modelling software architectures from various viewpoints. The architecture modelling capabilities of the UML have been influenced by this work. The ADL Acme [5] was used as the basis for the inclusion of architecture modelling capabilities into the UML [18, p. 203].

In this paper we present a precise mapping of the MidArch architecture modelling approach to the UML. MidArch was chosen because it is used in the application context of the MidArch design method (Section 2.1) and because of the similarity to ADLs such as Acme. When defining the mapping, we encountered several syntactic and semantic problems in incorporating ADL-oriented modelling constructs and the modelling constructs of the UML.

The UML provides a precise syntax for architectural concepts, but the semantics and pragmatics of their use exhibit many variation points. In order to define a specific operationalised UML-based modelling approach, these variation points must be specified. The OCL plays an important role in this specificaiton, as it is used to define precise conditions and restrictions for modelling software architectures consistently. The MidArch-to-UML/OCL mapping results in the definition of such a UML-based modelling approach, which can be used to model directly in the UML: the UML/MidArch Modelling Process.

An architectural style is a model of the structural aspects of a family of related software architectures. The UML does not natively provide a modelling construct comparable to architectural styles. Thus, UML/MidArch enables style-based architecture modelling with the UML in the first place. Style-based architecture modelling has the following benefits:

- It enables style conformance checks.
- It increases the conceptual integrity of architectural descriptions.
- The modelling of system architectures can be guided by the style, which results in greater modelling effi-

[1] By convention, the UML is not denoted an ADL within this paper.

ciency.

Compared with ADLs that provide a style modelling construct, UML/MidArch has the benefit that the underlying notation is more widely known, commercial tool support is available, and the integration of the component-and-connector view with other views in a single notation is possible.

In summary, the contributions of this paper are:

- The definition of a precise mapping of the MidArch modelling approach to the UML using OCL, which is largely consistent with both Acme and the UML and its underlying MOF metamodelling principles.
- The UML/MidArch Modelling Process that enables style-based modelling with the UML and OCL.
- An evaluation of the UML/MidArch Modelling Process through a small case study based on the pipes-and-filters architectural style.

Overview The remainder of the paper is structured as follows: First, we provide foundations in Section 2. Then we present the MidArch metamodel in Section 3. The design principles and the details of the mapping of the MidArch metamodel into the UML is presented in Section 4. Section 5 provides a case study that applies the resulting UML/MidArch approach. Section 6 reflects on the experience in defining and applying the modelling approach. In Section 7, related work is discussed, before Section 8 concludes the paper.

2. Foundations

In this section, we provide foundations regarding the research context in which the presented approach was developed (Sec. 2.1) and architectural style as a modelling construct (Sec. 2.2).

2.1 MidArch Method

The development of the modelling approach we present in this paper takes place in the context of the MidArch design method [6], which supports the systematic selection of a middleware platform. The MidArch method involves the definition of a target architecture based on the chosen middleware platform. A middleware platform is the combination of a middleware product [17] and a specific usage of that product. A core activity of the MidArch method is formal modelling of the architectural style of a middleware platform, i.e., the structural constraints imposed upon application architectures based on that platform. We refer to the result of this modelling as *Middleware-oriented Architectural Styles*, or MidArch Styles in brief.

In addition to serving as the unit of platform selection, MidArch Styles are used to guide modelling of a system architecture. In one project, candidate architectures are modelled and evaluated for several MidArch Styles. The evaluation results for the candidate architectures can be related to the MidArch Style and thus to the middleware platform. Therefore, MidArch allows reasoning on the quality properties of families of software architectures. This knowledge can be reused to improve the selection of a middleware platform in future similar project settings.

2.2 Architectural Style Modelling Construct

In conformance with the ISO/IEC DIS 25961 Standard [8] for architectural description, we use the term "(software) architecture" to refer to the fundamental organisation of an individual system. An "architectural style", on the other hand, characterises a family of related software architectures. More specifically, we consider the component-and-connector architectural viewpoint [2, ch. 3]. An architectural style, such as the pipes-and-filters or layered style, then characterises a family of configurations of components and connectors (in brief, a *configuration*).

An architectural style [11] consists of the definition of a vocabulary of component and connector metatypes, and of composition rules (constraints) for configurations. Architectural styles can be used to guide the development of a software architecture, since they constrain the vast architectural design space and thus ease design decisions. Conformance to an architectural style ensures internal conceptual coherence and consistency of an architecture. Moreover, it may establish specific quality properties of the resulting system, e.g. good scalability. Thus, checking an architecture for conformance to some architectural style is an important architectural analysis[2].

To facilitate such a *style conformance check*, notations are required that allow the specification of both architectural styles and architectures. Several ADLs that offer this feature have been developed, including Acme, Alfa, ArchWare and Wright.

Unfortunately, these notations and their underlying concepts 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 UML. Therefore, it would be beneficial to use the UML 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. However, the UML does not natively provide a modelling construct for architectural styles.

3. MidArch Metamodel

An overview of the MidArch Metamodel is given through Fig. 1.[3] It is partitioned by a dashed horizontal line, which

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

[3]In the text, we use the convention of using a sans-serif font for the concepts in the metamodel, e.g., Configuration.

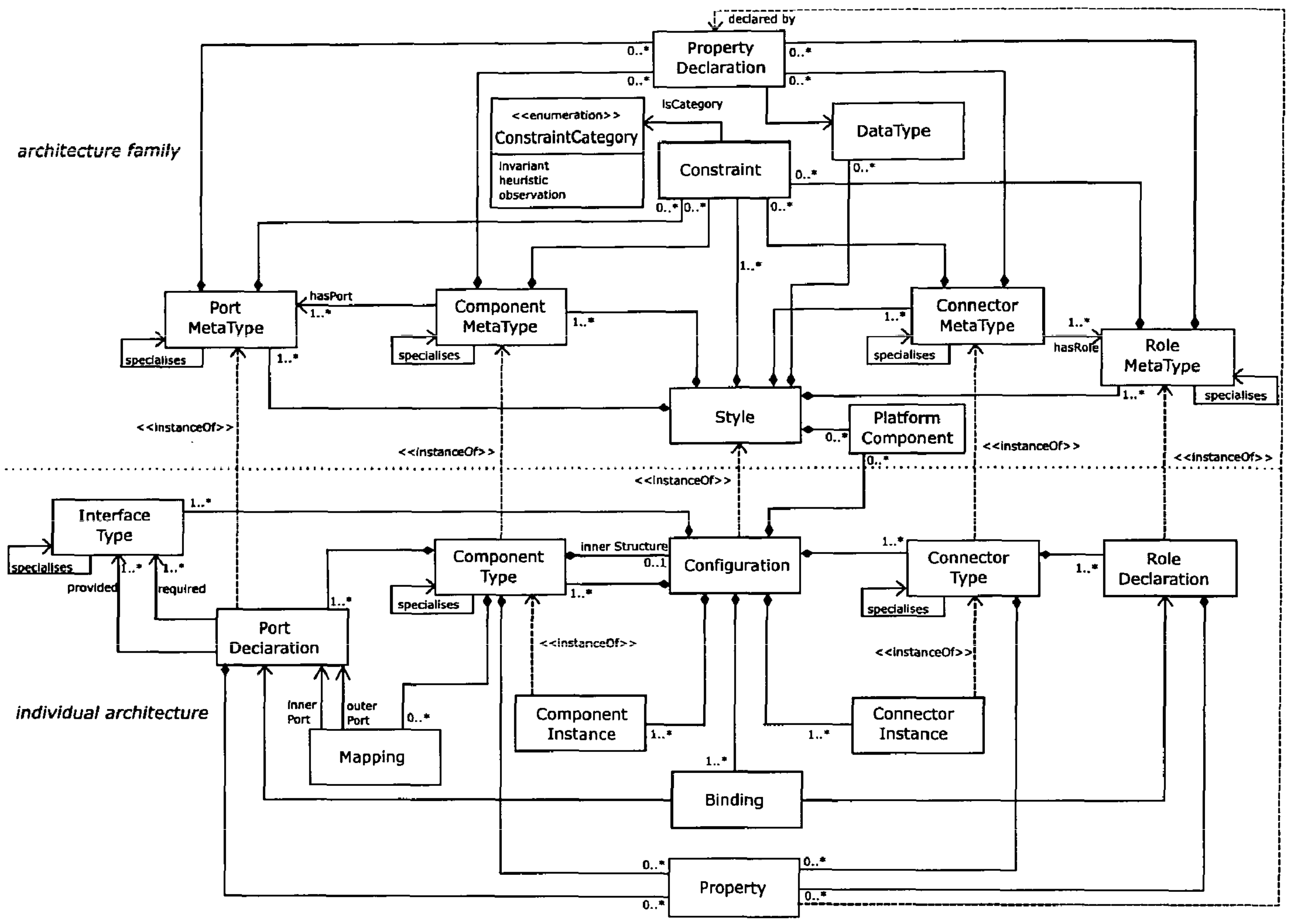


Figure 1. The MidArch Metamodel

separates the concepts concerning the style level (above the line) from the concepts concerning an individual architecture (below the line). We will briefly discuss both partitions in Sections 3.1 and 3.2.

3.1 Style Level

An architectural style, or in our terms a MidArch Style is at the core of the style modelling level. It has a unique name and 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.
- DataTypes are used for the definition of architectural properties.

3.2 System Level

A system is represented by a Configuration, which is declared to conform to exactly one Style. Application-specific types of components and connectors for an individual 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 component ports are bound to connector roles.

4. MidArch-to-UML/OCL Mapping

We use the convention of using small capitals for the metaclasses of the UML Superstructure [12, 13][4], e.g., COMPONENT.

As noted above, one important goal of the mapping is to enable style conformance checks. Another goal is to allow the reuse of UML tools, which restricts our modelling

[4]We essentially based our work on the UML 2.0 Specification, but consulted the newer UML 2.1.1 Specification in later phases of our work in case of ambiguities in the UML 2.0 Specification.

freedom to employing the lightweight extension mechanisms of the UML, i.e., specifying PROFILES containing STEREOTYPES and CONSTRAINTS.

Our approach builds upon the fundamental design choice to map each Style onto a separate UML PROFILE, and each Configuration conforming to a Style onto a UML model that primarily uses the metaclasses for the Component Diagram which applies the corresponding PROFILE (see Fig. 2). Thus, a Style is modelled essentially on the MOF M2 level, while Configurations are on the MOF M1 level. Due to the MOF metamodelling principles, they cannot be modelled entirely on the M2 level, as discussed in Sec. 6.

4.1 Design Principles

For each of the constructs of the MidArch metamodel, we investigated several mapping alternatives. These alternatives were gathered from the literature, or they were found through our own analysis of the UML Superstructure. The decision on the adopted mapping was subjected to several design principles to ensure the quality of the mapping:

- The mapping should retain the syntactic and semantic properties of each MidArch modelling construct.
- The mapping should retain the consistency between the MidArch modelling constructs. For example, ComponentMetaType must be mapped such that an instance of the UML target entity corresponds with ComponentType.
- The mapping should preserve the conceptual integrity of the UML.
- The model complexity of the target constructs should be as low as possible among the alternatives that conform with the other design principles (cf. [4, p. 32]), i.e., ideally, each MidArch modelling construct should be mapped to a single UML metaclass.

4.2 Mapping of the Modelling Constructs

Table 1 gives an overview of the mapping of the MidArch modelling constructs discussed in Sec. 3 to entities of the UML. It encompasses both the style and system levels.

For components and connectors, we start defining the mapping on the middle (i.e., type) level and then derive appropriate mappings for the metatype and instance levels. All metatype concepts are mapped to STEREOTYPES for the corresponding type respectively declaration construct.

MidArch Base Profile As an auxiliary means we defined a MidArch Base Profile, which is extended by each style profile. It serves to reduce the effort for modelling Styles and to harmonise the resulting PROFILES. The Base Profile is shown in Fig. 3. In addition to the STEREOTYPES for each of the modelling constructs the following OCL constraints are defined:

MidArch modelling construct	UML entity
Style	PROFILE
ComponentMetaType	STEREOTYPE for COMPONENTS
ComponentType	COMPONENT
ComponentInstance	INSTANCESPECIFICATION
Mapping	CONNECTOR (delegation)
PortMetaType	STEREOTYPE for PORTS
PortDeclaration	PORT within a COMPONENT
InterfaceType	INTERFACE
ConnectorMetaType	STEREOTYPE for CONNECTORS
ConnectorType	COLLABORATION
ConnectorInstance	COLLABORATIONUSE
RoleMetaType	STEREOTYPE for PORTS
RoleDeclaration	PORT within a COLLABORATION
Binding	DEPENDENCY
Constraint	CONSTRAINT
PropertyDeclaration	*tag definition*
Property	*tagged value*
PlatformComponent	COMPONENT
Configuration	PACKAGE
Instantiation relationship of Configurations and Styles	PROFILEAPPLICATION

Table 1. Mapping of MidArch modelling constructs to UML entities

recursiveComposition The parts of a MidArchComponent always are MidArchComponents again or MidArchConnectors.

useAlwaysPorts A MidArchComponent always interact through its ports.

onlyComponents A MidArchPort is always typed by a component.

onlyPortRoles The collaboration roles of a MidArchConnector are required to be MidArchRoles.

onlyPublicImports Only public imports are allowed.

importPlatform A package which apply a MidArch Style Profile has to import a package which apply a profile called AbstractPlatformLayer (not shown in this paper) which represents the provided services by the underlying platform.

Exemplarily, we discuss the rationale of the chosen mapping of ConnectorTypes and RoleTypes in detail.

Mapping Connectors and Roles While ComponentTypes can be mapped straightforwardly to COMPONENTS, more problems arise for ConnectorTypes. We investigated several options, before making the decision to map ConnectorTypes to COLLABORATIONS:

While the UML offers a CONNECTOR metaclass, it is not a first-class entity, as a connector is always owned by a CLASSIFIER (which is a superclass of COMPONENT). Moreover, connectors cannot be typed in the UML.

ASSOCIATION and INTERFACE are not suitable because they lack a means to specify roles.

CLASS and COMPONENT are more appropriate. In combination with them, INTERFACES can be used to describe RoleTypes. However, this mapping strategy still bears some problems: It conflicts with the idea to use pro-

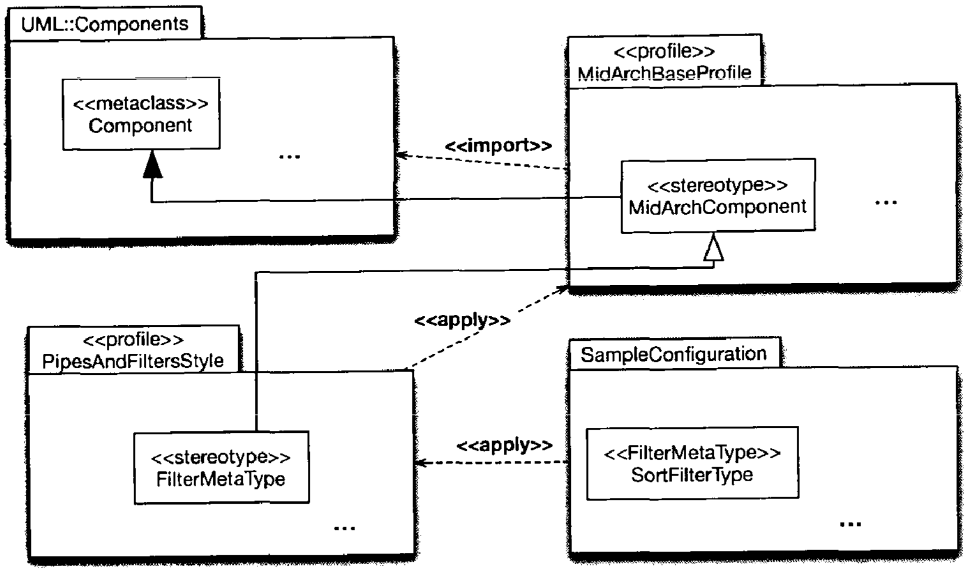

UML::Components
<<metaclass>> Component
...
<<import>>
<<profile>> MidArchBaseProfile
<<stereotype>> MidArchComponent
...
<<apply>>
<<profile>> PipesAndFiltersStyle
<<stereotype>> FilterMetaType
...
<<apply>>
SampleConfiguration
<<FilterMetaType>> SortFilterType
...

Figure 2. Relationships of the resulting UML/MidArch Models

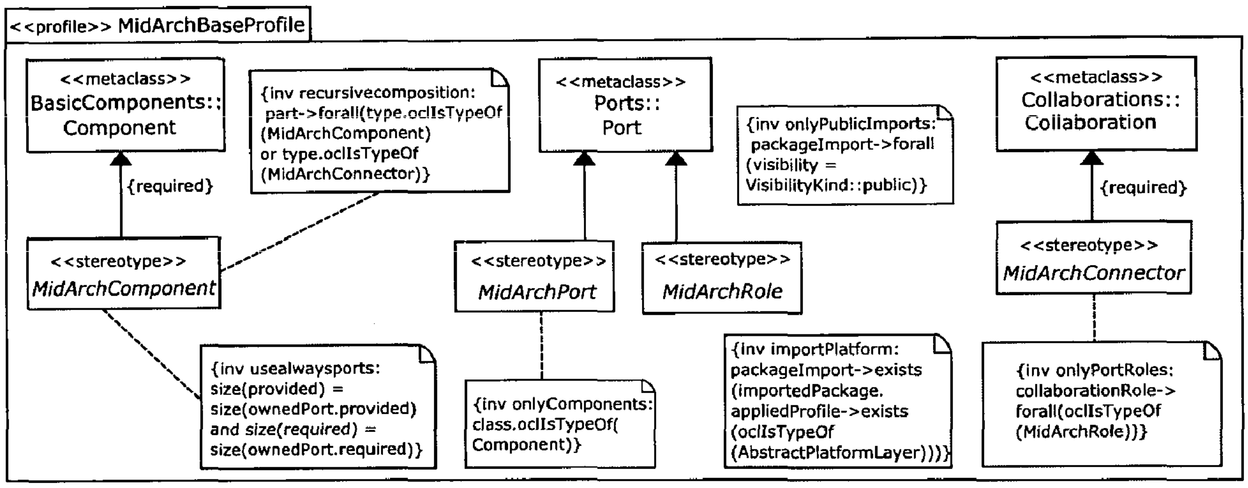

<<profile>> MidArchBaseProfile
<<metaclass>> BasicComponents:: Component
{required}
<<stereotype>> MidArchComponent
{inv recursivecomposition: part->forall(type.oclIsTypeOf (MidArchComponent) or type.oclIsTypeOf (MidArchConnector)}
{inv usealwaysports: size(provided) = size(ownedPort.provided) and size(required) = size(ownedPort.required)}
<<metaclass>> Ports:: Port
<<stereotype>> MidArchPort
<<stereotype>> MidArchRole
{inv onlyComponents: class.oclIsTypeOf(Component)}
{inv onlyPublicImports: packageImport->forall (visibility = VisibilityKind::public)}
{inv importPlatform: packageImport->exists (importedPackage. appliedProfile->exists (oclIsTypeOf (AbstractPlatformLayer)))}
<<metaclass>> Collaborations:: Collaboration
{required}
<<stereotype>> MidArchConnector
{inv onlyPortRoles: collaborationRole-> forall(oclIsTypeOf (MidArchRole))}

Figure 3. The MidArch Base Profile

vided and required INTERFACES for application-specific interfaces of ports and it is not possible to define two identical roles for the same ConnectorType. Furthermore, the use of COMPONENT for ConnectorTypes conflicts with the semantics of the UML.

Mapping ConnectorTypes to ASSOCIATIONCLASSES has the disadvantage of a high model complexity of the target structure, because additional ASSOCIATIONS are required to connect the roles of the connector (e.g., defined as PORTS) to the ports of components.

The metaclass COLLABORATION is best suited as target entity because roles can be defined through CONNECTABLEELEMENTS. Moreover, COLLABORATIONS can be typed because the metaclass inherits from CLASSIFIER.

Mapping Role Bindings As mentioned above, the mapping of ConnectorTypes to COLLABORATIONS implies the mapping of RoleDeclarations to CONNECTABLEELEMENTS. Since a role specifies the required set of features of a participating port and the metaclass PORT inherits from CONNECTABLEELEMENT, it is appropriate to use the metaclass PORT for describing roles.

Furthermore, the metaclass COLLABORATIONUSE serves as ConnectorInstance and provides a means to define role bindings with a set of DEPENDENCIES.

4.3 UML/MidArch Modelling Process for Styles

Now we are able to describe the UML/MidArch Modelling Process, which results from the mapping described in Sec. 4.2. The following steps must be taken in order to specify a style profile for a Style. These steps are applicable regardless of whether an existing Acme specification is translated to the UML, or the formalisation of the style is performed in the UML in the first place.

1. Create and denominate an empty PROFILE.
2. Add a PROFILEAPPLICATION to the MidArch Base Profile.
3. Import the abstract base stereotypes from the MidArch Base Profile.
4. Define STEREOTYPES for ComponentMetaTypes, PortMetaTypes, ConnectorMetaTypes and RoleMetaTypes which inherit from the respective abstract base stereotypes.
5. Define PropertyDeclarations as *tag definitions* of the defined STEREOTYPES.
6. Transcribe the style's Constraints into OCL and manifest as CONSTRAINTS.
7. Define additional OCL CONSTRAINTS to specify the associations of ComponentMetaTypes and their PortMetaTypes as well as ComponentMetaTypes and their RoleMetaTypes. This is necessary because it is not allowed to define associations between stereotypes when creating PROFILES.

5. Case Study

In this section, we describe a small case study in applying the UML/MidArch Modelling Process (Sec. 4.3) to the pipes-and-filters style. First we present the style itself, and then an example configuration that conforms to the style.

5.1 Example Style: Pipes-and-Filters

```
Family PipesAndFiltersFam = {
    Port Type inputT = {}
    Port Type outputT = {}

    Role Type sourceT = {}
    Role Type sinkT = {}

    Component Type Filter = {
        Port input : inputT =  new inputT
            extended with {};
        Port output : outputT =  new outputT
            extended with {};
        invariant Forall p : port in self.Ports
            |
            satisfiesType(p, inputT) OR
                satisfiesType(p, outputT)
      <<label : string = "Only␣ports␣of␣type␣
          inputT␣or␣outputT␣are␣allowed";>>;
    }

    Connector Type Pipe = {
        Role source : sourceT =  new sourceT
            extended with {};
        Role sink : sinkT =  new sinkT extended
            with {};
    }
}
```

Listing 1. A simplified Pipes-and-Filters Style in Acme

Listing 1 shows a simplified version of the pipes-and-filters style description that comes with AcmeStudio [16], which leaves out the initial and final `DataSource` and `DataSink` component metatypes as well as most constraints. The latter are only shown in the profile that resulted from the mapping shown in Fig. 4: It defines a connector metatype PIPEMETATYPE with roles SOURCET and SINKT, as well as a component metatype FILTERMETATYPE with ports INPUTT and OUTPUTT. The following OCL constraints are defined:

portType FILTERMETATYPE components may only have INPUTT or OUTPUTT ports.

existIn/existOut FILTERMETATYPE components have at least one INPUTT resp. OUTPUTT port.

pufferSize The buffer size of a PIPEMETATYPE connector must be at least 0.

twoRoles A PIPEMETATYPE connector has exactly two roles.

existSource/existSink A PIPEMETATYPE connector has at least one SOURCET resp. SINKT role.

noDanglingRoles A PIPEMETATYPE connector has no unconnected roles.

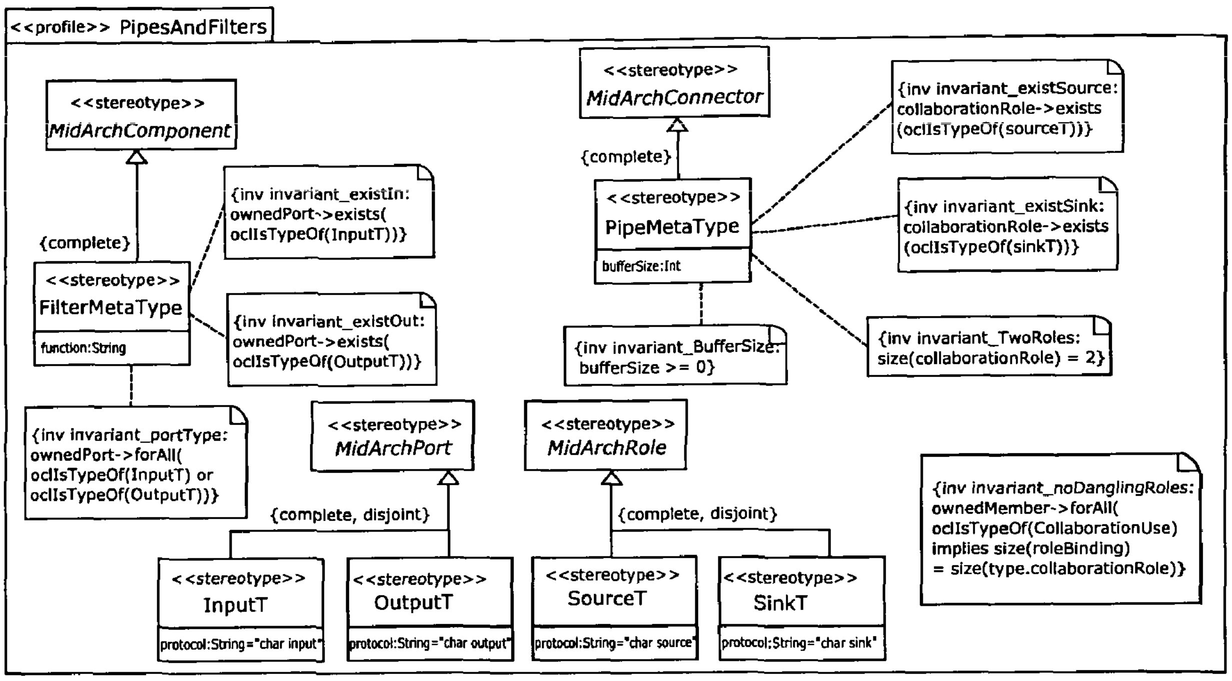
5.2 Example Configuration in the Pipes-and-Filters Style

Fig. 5 shows a UML package containing a example configuration that is member of the pipes-and-filters style family. It declares two filter types (upper left part) and one pipe type (upper right part) including their ports and roles and default values of their architectural properties. Finally these types are used in a small configuration (lower part) that demonstrates how to bind ports and roles within our approach.

6. Evaluation

6.1 Mapping Definition

Basic Approach The chosen approach to map Styles to PROFILES is the only lightweight possibility to achieve the requirement of enabling style conformance checks with UML/OCL tools such as those required for MDA approaches. There have been proposals for modelling design patterns as M1-level UML models, which cannot be used to model styles, since their instantiation cannot be specified within the UML. If one chose some metaclass X to represent styles, configurations would need to be represented as INSTANCESPECIFICATIONS referring to instances of X. The semantics of these INSTANCESPECIFICATIONS cannot be specified in the UML specifically for some referenced metaclass. While the UML would allow an extension with user-defined semantics of INSTANCESPECIFICATIONS, standard UML/OCL tools would not be able to interpret these semantics and perform the style conformance check



<<profile>> PipesAndFilters
<<stereotype>> MidArchComponent
{complete}
<<stereotype>> FilterMetaType
function:String
{inv invariant_existIn: ownedPort->exists(oclIsTypeOf(InputT))}
{inv invariant_existOut: ownedPort->exists(oclIsTypeOf(OutputT))}
{inv invariant_portType: ownedPort->forAll(oclIsTypeOf(InputT) or oclIsTypeOf(OutputT))}
<<stereotype>> MidArchConnector
{complete}
<<stereotype>> PipeMetaType
bufferSize:Int
{inv invariant_BufferSize: bufferSize >= 0}
{inv invariant_existSource: collaborationRole->exists (oclIsTypeOf(sourceT))}
{inv invariant_existSink: collaborationRole->exists (oclIsTypeOf(sinkT))}
{inv invariant_TwoRoles: size(collaborationRole) = 2}
<<stereotype>> MidArchPort
{complete, disjoint}
<<stereotype>> InputT
protocol:String="char input"
<<stereotype>> OutputT
protocol:String="char output"
<<stereotype>> MidArchRole
{complete, disjoint}
<<stereotype>> SourceT
protocol:String="char source"
<<stereotype>> SinkT
protocol:String="char sink"
{inv invariant_noDanglingRoles: ownedMember->forAll(oclIsTypeOf(CollaborationUse) implies size(roleBinding) = size(type.collaborationRole)}

Figure 4. A Profile for the Pipes-and-Filters Style

Implicit representation of metatype relationships Since the UML does not allow the specification of ASSOCIATIONS between STEREOTYPES, any relationships between metatypes, which are explicit in the metamodel, can only be represented indirectly through CONSTRAINTS that are associated with the respective STEREOTYPES. These CONSTRAINTS are specified on the M2 level, and indirectly require the existence of certain ASSOCIATIONS on the M1 level.

Therefore, a UML/OCL mapping of a Style contains more CONSTRAINTS than the original style Constraints. These could be distinguished by applying some naming convention.

6.2 Case Study

Complexity For abstract architectural styles, such as the pipes-and-filters style from the case study, which define only a small vocabulary and few constraints, the approach is applicable well. It produces profiles whose size is manageable. We also modelled more complex middleware-oriented architectural styles (e.g., for the Avalon Framework [6]). For these, the profiles are more complex, and difficult to arrange within a single diagram. However, this is a general challenge not only of the UML but of any graphical modelling notation.

Representation of Allowed Port/Role Binding When Acme family specifications make explicit statements on the allowed binding of ports and roles on the metatype level, these can only be translated indirectly to UML/MidArch, as already noted in Sec. 6.1. In part, this is due to a variation point in Acme. The MidArch metamodel assumes that RoleMetaTypes declaratively specify conditions concerning which PortMetaTypes may connect to them. In addition, it assumes implicitly that a Port's required Interfaces must be provided by the Port at the other end of the attached Connector, which is not required in Acme. Since this relationship is only specified at the system level, it can be explicitly expressed in the M1 UML model.

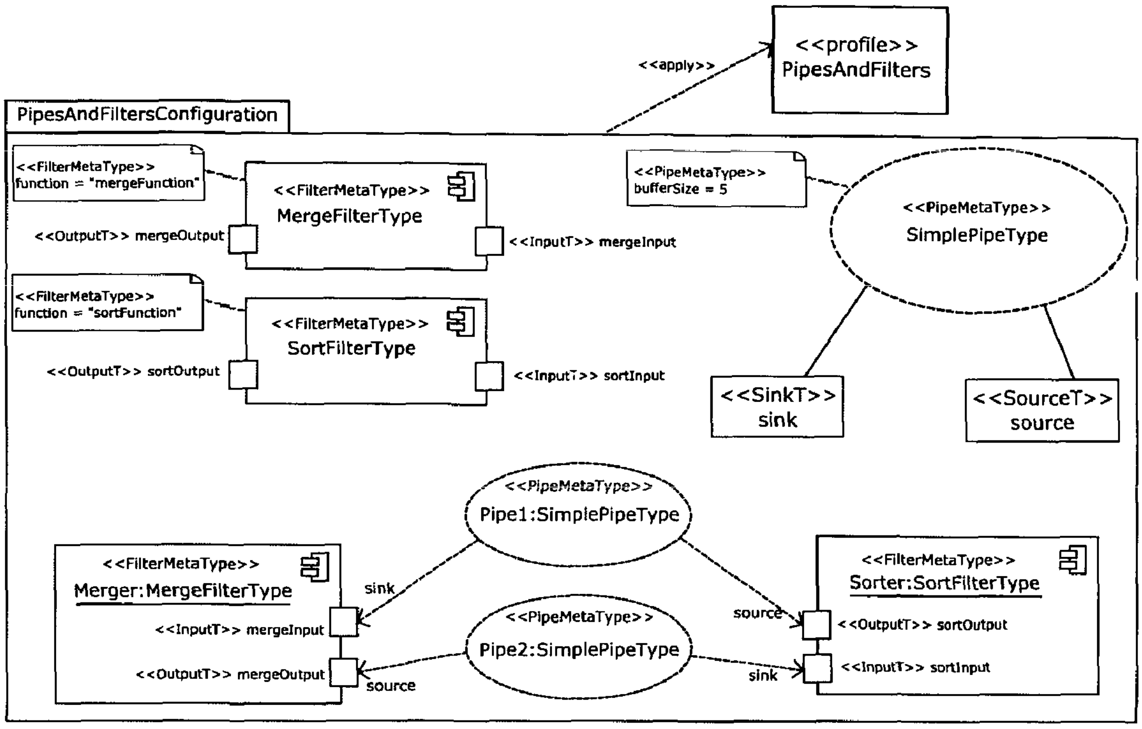
7. Related Work

To our knowledge, there is no prior work that proposes a platform-independent approach to model styles and style-based architectures in the UML.

UML Profile-based Architectural Modelling Approaches Some authors have proposed *generic* profiles for modelling software architectures in the UML (e.g., [9]), but they do not consider style-specific profiles.

Roh et al. [15] pursue a similar approach as we do: They define a domain-independent base profile and domain-specific profiles that specialise the former. However, their profiles are not correct UML profiles: They define associations between STEREOTYPES, which is not allowed. It remains unspecified how the COLLABORATIONS they define are instantiated to COLLABORATIONUSES. A similar problem also applies to [3].

Selonen et al. [19] adopt the concept of UML profiles for a mapping of a set of architectural views (e.g., structural view and behavior view) to a taxonomy of related profiles.



<<profile>> PipesAndFilters
<<apply>>
PipesAndFiltersConfiguration
<<FilterMetaType>> function = "mergeFunction"
<<FilterMetaType>> MergeFilterType
<<OutputT>> mergeOutput
<<InputT>> mergeInput
<<FilterMetaType>> function = "sortFunction"
<<FilterMetaType>> SortFilterType
<<OutputT>> sortOutput
<<InputT>> sortInput
<<PipeMetaType>> bufferSize = 5
<<PipeMetaType>> SimplePipeType
<<SinkT>> sink
<<SourceT>> source
<<PipeMetaType>> Pipe1:SimplePipeType
<<FilterMetaType>> Merger:MergeFilterType
<<InputT>> mergeInput
<<OutputT>> mergeOutput
sink
source
<<PipeMetaType>> Pipe2:SimplePipeType
<<FilterMetaType>> Sorter:SortFilterType
<<OutputT>> sortOutput
<<InputT>> sortInput
source
sink

Figure 5. An example configuration in the Pipes-and-Filters Style

The approach of Baresi et. al. [1] is similar to our concept concerning their design of a metamodel for architectural description. In essence they propose to define UML profiles for platform-specific models (e.g., a SOA profile).

Zarras et al. [21] discuss the design of a base profile for architectural description with the UML 1.3. They mention the possibility to use UML profiles for the description of architectural styles, but do not present a detailed approach.

Garlan et al. [4] present several mapping alternatives to translate the basic modelling constructs of the component-and-connector view to UML metaclasses. Similarly to our approach, they define criteria (e.g., semantic match and visual clarity) to make a selection.

Mapping ADLs into the UML There have been other approaches to map ADLs to the UML, such as [7, 10, 14], but they do not address the problem of describing styles and style-based architectures.

8. Conclusion

We have defined a precise mapping of the Acme-based MidArch metamodel for architectural description to the UML 2, and have defined a method for modelling architectural styles and style-based architectures on the basis of the UML 2. This indicates that the UML 2 in combination with OCL is suited for modelling the component-and-connector view of software architectures. The advancement of the COMPONENT metaclass and the introduction of the new PORT metaclass have significantly improved UML's capabilities to model component-and-connector architectural views compared to the UML 1.x.

We have demonstrated the feasibility of style-based architecture modelling using UML/MidArch by a small case study. We have already done some more complex case studies, which have also shown the feasibility of the approach, while there are still some messy points due to the MOF metamodelling principles underlying the UML (see Sec. 6).

The mapping could be adapted to the original Acme language with only slight modifications, so that an automatic translation from Acme to the UML-based modelling approach and vice versa should be implementable, since the deviations to MidArch are only minor. This could be done either through XSL transformations between xAcme and XMI, or through model-to-model transformations between a MOF-based representation of Acme and the UML.

However, as a result of the mapping we defined the UML/MidArch Modelling Process for modelling styles and style-based component-and-connector configurations, which is of practical relevance on its own. It forms the basis of style evaluations in the context of the MidArch Method (see Sec. 2.1).

Specialisation relationships between styles are of great relevance to the MidArch Method. These have not been addressed in this paper, but they are already defined in the MidArch metamodel and the mapping. Further refinement is necessary.

Opportunities for future work include the incorporation of modelling heterogeneous architectures that have multiple styles, and of style-specific design patterns [11] and their instances. While generic UML tools can be used for the approach if they support custom UML PROFILES, specific tool support for using the style to guide the definition of system architectures would be beneficial. A UML/MidArch PROFILE is a form of an explicit platform model [20], which could be used within PIM to PSM transformations. This use should be further investigated. We are working on implementation mappings for the MidArch metamodel for the Spring framework, which are currently bound to Acme representations. The implementation mappings should be integrated with UML/MidArch as well.

References

[1] L. Baresi, R. Heckel, S. Thöne, and D. Varró. Style-based modeling and refinement of service-oriented architectures. *Software and Systems Modeling*, 5(2):187–207, June 2006.

[2] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond.* Addison-Wesley, Boston, 2002.

[3] L. Fuentes, J. M. Troya, and A. Vallecillo. Using UML Profiles for documenting web-based application frameworks. *Ann. Software Eng.*, 13(1-4):249–264, 2002.

[4] D. Garlan, S.-W. Cheng, and A. J. Kompanek. Reconciling the needs of architectural description with object-modeling notations. *Sci. Comput. Program.*, 44(1):23–49, 2002.

[5] D. Garlan, R. Monroe, and D. Wile. Acme: Architectural description of component-based systems. In G. Leavens and M. Sitaraman, editors, *Foundations of Component-Based Systems*, pages 47–68. Cambridge University Press, 2000.

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

[7] P. Inverardi, H. Muccini, and P. Pelliccione. DUALLY: Putting in synergy UML 2.0 and ADLs. In *WICSA '05: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 251–252. IEEE Computer Society, 2005.

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

[9] M. M. Kandé and A. Strohmeier. Towards a UML Profile for software architecture descriptions. In *Proc. UML 2000*, volume 1939 of *Lecture Notes in Computer Science*, pages 513–527. Springer, 2000.

[10] N. Medvidovic, D. Rosenblum, D. Redmiles, and J. Robbins. Modeling software architectures in the Unified Modeling Language. *ACM Transactions on Software Engineering and Methodology*, 11(1):2–57, 2002.

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

[12] OMG. UML 2.0 Superstructure Specification, 2006. `www.omg.org/docs/formal/05-07-04.pdf`.

[13] OMG. UML 2.1.1 Superstructure Specification, 2007. `www.omg.org/docs/formal/07-02-03.pdf`.

[14] F. Oquendo. Formally modelling software architectures with the UML 2.0 Profile for π-ADL. *SIGSOFT Softw. Eng. Notes*, 31(1):1–13, 2006.

[15] S. Roh, K. Kim, and T. Jeon. Architecture Modeling Language based on UML2.0. In *APSEC '04: Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*, pages 663–669, Washington, DC, USA, 2004. IEEE Computer Society.

[16] B. Schmerl and D. Garlan. AcmeStudio: Supporting style-centered architecture development. In *Proc. ICSE '04*, pages 704–705, Washington, DC, USA, 2004. IEEE Computer Society.

[17] D. C. Schmidt and F. Buschmann. Patterns, frameworks, and middleware: their synergistic relationships. In *Proc. ICSE '03*, pages 694–704, Washington, DC, USA, 2003. IEEE Computer Society.

[18] B. Selic. Architectural modeling capabilities of UML 2.0. In C. Rupp, J. Hahn, S. Queins, M. Jeckle, and B. Zengler, editors, *UML 2 glasklar*, chapter 9. Carl Hanser, München, 2005.

[19] P. Selonen and J. Xu. Validating UML models against architectural profiles. In *Proceedings of the 9th ESEC/FSE-11, Helsinki*, pages 58–67, New York, NY, USA, 2003. ACM Press.

[20] D. Wagelaar and V. Jonckers. Explicit Platform Models for MDA. In L. C. Briand and C. Williams, editors, *Proceedings of Model Driven Engineering Languages and Systems, 8th (MoDELS 2005)*, volume 3713 of *Lecture Notes in Computer Science*, pages 367–381. Springer, 2005.

[21] A. Zarras, V. Issarny, C. Kloukinas, and V. Nguyen. Towards a Base UML Profile for Architecture Description. In *Proceedings of ICSE 2001 Workshop for Describing Software Architecture with UML*, pages 22–26. IEEE Computer Society, 2001.