Style-Based Software Architectural Compositions as Domain-Specific Models

Nikunj R. Mehta, Ramakrishna Soma, Nenad Medvidovic
Department of Computer Science
University of Southern California
941 W. 37th Pl., Los Angeles, CA, 90089-0781, USA
{mehta | rsoma | neno}@usc.edu

Abstract

Architectural styles represent composition patterns and constraints at the software architectural level and are targeted at families of systems with shared characteristics. While both style-specific and style-neutral modeling environments for software architectures exist, creation of such environments is expensive and frequently involves reinventing the wheel. This paper describes the rapid design of a style-neutral architectural modeling environment, ViSAC. ViSAC is a domain-specific modeling environment obtained by configuring Vanderbilt University's Generic Modeling Environment (GME) for Alfa, a framework for constructing style-based software architectures from architectural primitives. Users can define their own styles in ViSAC and, in turn, use them to design software architectures. Moreover, ViSAC supports the hierarchical design of heterogeneous software architectures, i.e., using multiple styles. The rich user interface of GME and support for domain-specific semantics enable interactive design of well-formed styles and architectures.

1. Introduction

Architectural styles codify the recurring design practices and successful organizations of software systems. Styles are the composition patterns of and constraints on the computing, interaction, and data elements of the architectures of similar software systems [24]. Styles are considered useful for instituting high-level reuse and bringing economy to the design of software architectures [19]. Modern software systems are often composed *hierarchically*, and frequently use *multiple* styles in an architecture [14]. A number of styles have emerged in research and industry, such as publish-subscribe, client-server, peer-to-peer, and so on. A prominent reason for the increasing use of styles in the design of software architectures is the observation that a property proven about a style holds true in the architectures based on that style [9]. Further, styles provide analytical models for determining properties of software architectures such as performance and reliability [8, 27].

In order to support the growing use of styles in software development, there is a pressing need for software engineering techniques and tools that support systematic style-based architectural design. Past research has helped establish the utility of architectural styles for designing large-scale software systems by providing style-specific environments for architectural design (e.g., DRADEL [15]) and style-based architectural description languages (e.g., Weaves [8]). The creation of such techniques and associated tools is highly expensive, time consuming, and frequently involves reinventing the wheel. Moreover, such approaches do not take into account the basic similarities among architectural styles. More recent research has focused on style-neutral design environments (e.g., AcmeStudio [25]) and architectural description languages (e.g., xADL [3]). These generic approaches support user-defined styles and, therefore, eliminate the need to create new infrastructure and environments for every new style. Still, such approaches do not recognize the architectural primitives underlying different styles and style-based software architectures (architectures as a shorthand). The knowledge of such primitives enable an architect to better understand the relationships between architectural styles.

In the context of the Alfa project, our research is aimed at supporting the systematic construction of architectures for network-based systems from architectural primitives [16]. The Alfa framework provides a small set of architectural primitives and formalizes a theory for composing style and software architectures [17]. This theory accounts for hierarchical composition of architectural elements and supports heterogeneous architectures, i.e., using multiple styles. This paper describes the rapid design of a style-neutral architectural modeling environment, *ViSAC* for Alfa. ViSAC is a domain-specific modeling environment obtained by configuring Vanderbilt University's Generic Modeling Environment (GME) [11].

ViSAC is designed as a domain-specific modeling environment based on the composition theory and architectural primitives of Alfa. Domain-specific design environments, such as Matlab/Simulink [13], have been considered useful for capturing specifications in specific engineering fields. Modeling environments that can be tailored for use in specific domains support the specification of custom meta models for tailoring the environments for use in those

domains. ViSAC is specified as a custom meta model for such a configurable modeling environment, GME. This allows us to leverage existing investments in a design environment and enables the rapid development of a rich user interface for visually modeling styles and architectures. GME is customized through the use of its meta modeling notation for defining a visual language for Alfa, i.e., *xAlfa*. ViSAC takes advantage of the semantic analysis features of GME to check well-formedness of models and, therefore, supports interactive design of styles and style-based as well as style-less architectures.

ViSAC has been successfully applied to styles for network-based systems. In this paper, we illustrate the design of the software architecture of a web-based photo album system for viewing and managing images. ViSAC interactively helps prevent errors of malformations in models of styles and architectures. Hierarchical composition is supported by the concept of model hierarchy in GME. Moreover, GME's rich drag-and-drop support and type inheritance mechanisms simplify model design and enable reuse of existing xAlfa models.

The rest of this paper is organized as follows. Section 2 provides background material on the Alfa framework and GME. Section 3 discusses our approach for designing and well-formedness analysis of architectures through domain-specific modeling techniques and GME. Section 4 illustrates the use of ViSAC in the design of an example heterogeneous architecture. Section 5 discusses the strengths and limitations of our approach. Section 6 discusses related work in the areas of domain-specific modeling, visual languages, and software architectures. Finally, Section 7 presents some conclusions of our approach and pointers to future work.

2. Background

In this section we provide a brief summary of the Alfa framework and describe the concepts of modeling in GME to serve as the background for rapidly designing a modeling environment for styles and architectures.

2.1. Alfa

Alfa is a framework for understanding and constructing style-based architectures from a small set of architectural primitives [16]. Alfa's eighteen architectural primitives are classified along five orthogonal characteristics of styles: structure, behavior, interaction, topology, and data.

Alfa employs point-to-point communication channels, called *ducts*, to tie together computational elements of a style. In Alfa, both software components and connectors can be composed hierarchically; at the leaf nodes of this hierarchy are primitive components and connectors. Alfa's primitives consist of nine nouns, capturing the *form* of architectural style elements, and nine verbs capturing the elements' *function* [17]:

(1) Data – DATUM
(2) Structure – PARTICLE, OUTPUT, INPUT, INTERFACE, TWOWAY
(3) Interaction – DUCT, RELAY, BIRELAY, HOLDS, LOSES
(4) Behavior – SEND, RECEIVE, HANDLE, REPLY
(5) Topology – CREATE, CONNECT, DISCONNECT

The composition of styles and architectures discussed in this paper is based on Alfa's composition theory [17], which also provides additional details about Alfa's architectural primitives. Architectural primitives and their interrelationships are formalized in this theory to support the modeling of styles and architectures. This theory supports heterogenous architectures, i.e., using multiple styles, and enables data type checking in styles and software architectures. In this theory, styles provide templates, i.e., parameterized types, that are refined as architectural instances. Alfa's composition theory defines the refinement relationships from style templates to architectural instances along the five style dimensions.

2.2. GME concepts

GME is a configurable modeling environment that supports the easy creation of domain-specific modeling environments (DSMEs) [11]. DSMEs are considered useful for capturing specifications in specific engineering fields. Large scale, complex models can be built in GME based on a few underlying concepts. These concepts are: hierarchy, multiple aspects, meta modeling objects, and semantic constraints. These concepts manifest themselves as the meta model elements in GME, expounded in [7]:

- *Models* are compound objects in GME, which may have parts and inner structure to support hierarchical composition.
- *Atoms* are elementary objects that cannot contain objects.
- *Sets* are used to specify groups of objects.
- *References* are used as placeholders for objects
- *Connections* are used to express relationships among the elements of a model.
- Atoms, models, sets, and connections may have *attributes* for recording textual information.
- *Aspects* are the means to control visibility of model objects. A model may have multiple aspects, with only a subset of its parts and their relationships visible in each aspect. This is aimed at reducing view complexity and aiding understandability.
- *Projects* and *folders* are containers used to organize models.

Every model is a type. Types can be s*ub-typed* and *instantiated* in GME, both of which produce deep copies that are linked to their types. In order to prevent multiple dependencies, types can only be derived or instantiated if they do not themselves contain sub-types or instances. Sub-types can be modified to add new parts. However, existing parts cannot be removed in either sub-types or type instances.

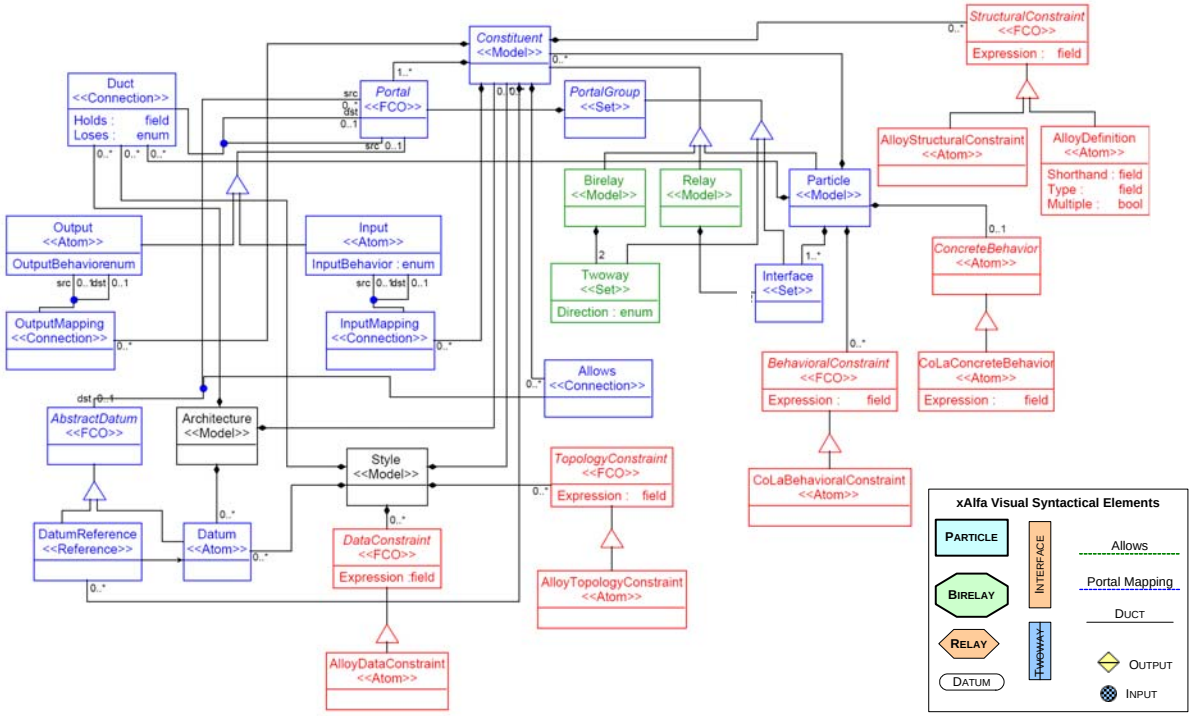


Figure 1. Class diagram of xAlfa paradigm in GME

A DSME is designed in GME by specifying a modeling *paradigm* through a process called *meta modeling*. Such a paradigm defines a language for modeling systems in a specific domain. DSME paradigms are specified in GME using GME's *meta modeling* paradigm. The meta modeling paradigm consists of four different views: class diagram, aspects, constraints, and attributes. The UML class diagram notation, extended with a few constructs, is used for designing DSME paradigms.

During meta modeling, the *syntax* of the modeling language is defined as the valid entities and relationships in the domain, and *semantics* of the modeling language are defined through constraints on the elements of the meta model [12]. Graphical icons are also associated with each valid entity, and various kinds of lines and line ends are associated with each relationship. Semantics or the rules that specify the well-formedness of the domain models are specified using MCL, a language based on the Object Constraint Language (OCL) 1.4 [21].

GME's constraint checking engine evaluates paradigm-specific semantics on user created models at run-time and flags any errors. Additionally, GME makes provisions for selecting events, (e.g., when the element is created, when the current model is closed, etc.) that trigger the constraint to be checked. If no event is chosen to trigger checking of a constraint, then the constraint is checked only when the constraint manager is explicitly invoked.

Every paradigm can be associated with one or more interpreters that analyze domain-specific models in that paradigm. GME's meta modeling paradigm provides one such interpreter that generates a GME configuration for paradigms specified in it. This configuration is the visual language that includes the syntactic and semantic rules specified by the paradigm designer.

3. A Domain-Specific Modeling Environment for Alfa

ViSAC, a DSME for styles and architectures, is designed using GME by specifying a paradigm for Alfa, called xAlfa. This section describes the syntactic and semantic elements of xAlfa, which will be used later to define architectural styles and compose architectures.

The systematic composition of style and architectural elements using Alfa's composition theory and its primitives requires a precise notation and a design environment to manipulate models using the notation. Our main objective in the design of such a notation and environment is to facilitate interaction during the composition of architectures. To this end, we have designed a visual modeling environment using GME. A domain-specific modeling paradigm for modeling Alfa compositions, called xAlfa, is specified in the meta modeling paradigm as shown in Figure 1. xAlfa reifies Alfa's composition theory and primitives in the form of various GME meta model objects.

Entity specification. The first step in specifying the xAlfa paradigm is the identification of GME meta model objects for Alfa's primitives. As shown in Figure 1, primitives that correspond to the composite terms of Alfa's composition theory are treated as GME models. For example, both styles and architectures are top-level models. Also, a

constituent is a model as it contains portals and interfaces. GME atoms are used for non-composites in the style template hierarchy, i.e. for DATUMs and portals. Moreover, GME sets are used to model partitions of portals such as INTERFACEs. GME references are used for referencing DATUMs within a constituent. Various user-specified stylistic constraints, such as *AlloyStructuralConstraint*, are also defined as atoms. Further, a connection object is used for DUCT. The allowed DATUM of portals and portal mappings are also modeled as connections. The visual representation of some of xAlfa's syntactical elements is also shown in Figure 1.

Relationship specification. The second step in specifying xAlfa requires the identification of the relationships among the objects of xAlfa. Composition relationships are created according to Alfa's composition theory. For example, a style composes constituents, and a constituent composes portals. Cardinality of the composition relationship is also specified along with the role played by the composed model in the composition relationship. Similarly the cardinality of association relationships for connection objects is also specified as per Alfa's composition theory. For example, any portal may be associated in at most one DUCT with another portal.

Reference relationships are created between the referring object and referred object. For example, *Datum Reference* refers to *Datum*. Inheritance relationships are used to indicate concrete implementations of abstract kinds. The use of abstract meta model objects reduces the repetition of relationships across concrete objects derived from them. For example, PARTICLE, RELAY, and BIRELAY can each be modeled as concrete constituents, such that their common relationships can be housed in constituents. Inheritance relations are also useful to support evolution as in the case of various stylistic expressions such as *StructuralConstraint* and *ConcreteBehavior*.

Attribute specification. The third step of designing this meta model involves the identification of attributes of the meta model objects. Three kinds of attributes are available, enumerations, integers and fields. For example, HOLDS and LOSES are recorded as attributes of a DUCT. Whereas HOLDS is specified as an integer, LOSES is specified as an enumeration. The enumeration choices are specified according to Alfa's composition theory. A field attribute, called expression, is used to record notation-specific details of a stylistic constraint.

Constraint Specification. Specification of semantic rules of well-formedness in xAlfa enables checking styles and architectures for malformations. These semantic rules are adapted from Alfa's composition theory [17] to GME's constraint specification notation, MCL. The meta model objects and their relationships are used for writing constraints. GME also defines a library of GME objects available during constraint checking. GME also defines a library of OCL objects available to the GME constraint manager. As examples, two constraints corresponding to two rules of Alfa are shown below.

Rule:
All allowed datums of portals are defined in the style.

MCL constraint on DatumReference:

```
let style = self.refersTo().parent().oclAsType(
Style) in
 let models = style.models() in
      models->select(m:Model | m.referenceParts(
      DatumReference).includes(self))->notEmpty()
```

Rule:

Portals should be partitioned into disjoint groups of INTERFACEs and TWOWAYs

MCL constraint on Portal:

```
self.memberOfSets(Interface)->size = 1 or
self.memberOfSets(Twoway)->size = 1
```

In xAlfa, the choice of triggering constraint verification is aimed at providing the earliest possible feedback to the architect. As a result, the first rule is setup for verification upon user command, whereas the second rule is checked when a model is closed, and when a portal is included or excluded from a set.

Aspect Specification. The final step in specifying xAlfa is identifying the aspect(s) associated with each meta model object. xAlfa defines three aspects: *Interface*, *Composition*, and *Constraint*. The definition of portals, their grouping into INTERFACEs and TWOWAYs, and their allowed DATUMs are all performed in the Interface aspect. The hierarchical composition of constituents as well as the specification of DUCTs and portal mapping, is performed in the Composition aspect. Finally, all the stylistic constraints are defined in the Constraint aspect.

Once the xAlfa paradigm is specified completely, the meta modeling interpreter produces a configuration for GME that supports domain-specific modeling of styles and architectures. When this configuration is loaded into GME, we get ViSAC, as shown in Figure 2, customized for Alfa.

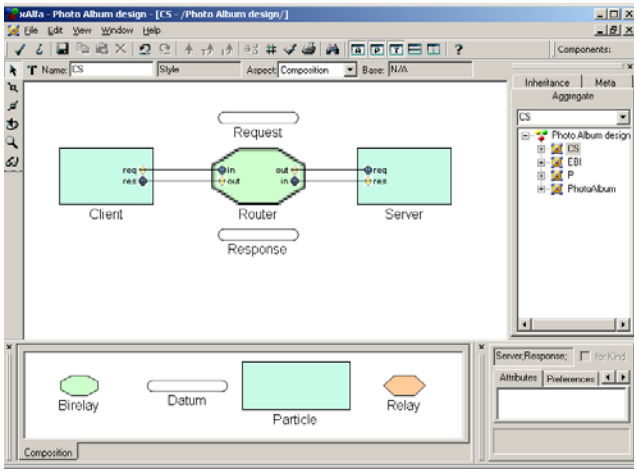


Figure 2. ViSAC showing client-server style

4. Modeling Styles and Architectures

To illustrate our approach of designing styles and architectures, we consider the software architecture of a web-based photo album system, shown informally as a box-and-line diagram in Figure 3. This architecture employs hierarchical composition as well as multiple styles.

This architecture comprises two types of clients: *image manager* for managing images, and *image viewer* for viewing images. Further, an *album server* stores and retrieves images, and, optionally, enhances images being stored. Multiple clients are connected to an album server over a *network*. Image viewer provides a visual interface for choosing an image to be displayed, and for displaying a chosen image. Image manager clients consist of a *manager* and a *manager interface* that communicate via *events*. Internal components of an image manager are used to select images to be stored on the server, as well as to select enhancements to be performed on a selected image. The album server also internally contains image *decoding*, *enhancing*, and *encoding* filters connected via *pipe*s.

This software architecture employs three different architectural styles—client-server (CS), pipeline (P), and event-based integration (EBI)—and involves hierarchical composition of its elements. The composition of this software system's architecture using Alfa's primitives consists of two phases: style definition and architecture definition. In the first phase, we define the three styles used in the software architecture. Each of the styles is recorded as a top-level model. For example, two levels of composition of the pipeline style are shown in Figure 4a using the visual syntactical elements shown in Figure 1. The definition of portals, interfaces and allowed datums takes place in the Interface aspect as shown in Figure 4b. Figure 5 shows two levels of the composition of the event-based integration style. Earlier, Figure 2 showed the composition of the client/server style. To illustrate the results of run-time constraint checking, Figure 6 shows a constraint violation error in the EBI style. This error occurs when the rule on disjoint partition of portals as described in the previous section is violated.

As the second step, required data types and constituents, i.e., components and connectors, in the software architecture are refined from stylistic templates. Figure 7 shows two levels of hierarchy of the architecture from Figure 3, where the top level is client-server style (Figure 7a), and the lower levels use pipe-and-filter (Figure 7b) and event-based integration (Figure 7c) styles.

The refinement of constituent templates using GME sub-typing is made difficult by the sub-typing rules of GME. For example, *Store Image* data type is obtained by sub-typing the *Request* datum. As a datum is an atom, its sub-typing is always allowed by GME. Sub-typing results in duplication of all the parts of a type, none of which can be deleted. Moreover, it is not possible to duplicate parts of a model in its sub types. All these sub-typing rules conflict with our needs for refining constituent templates. For

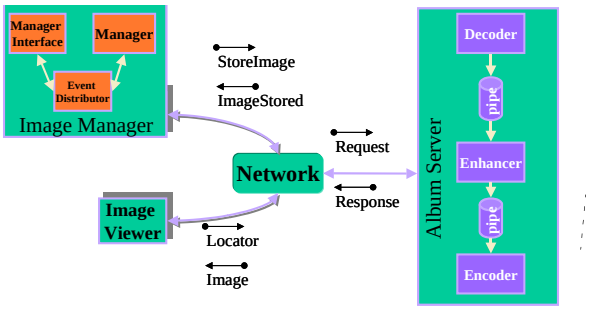
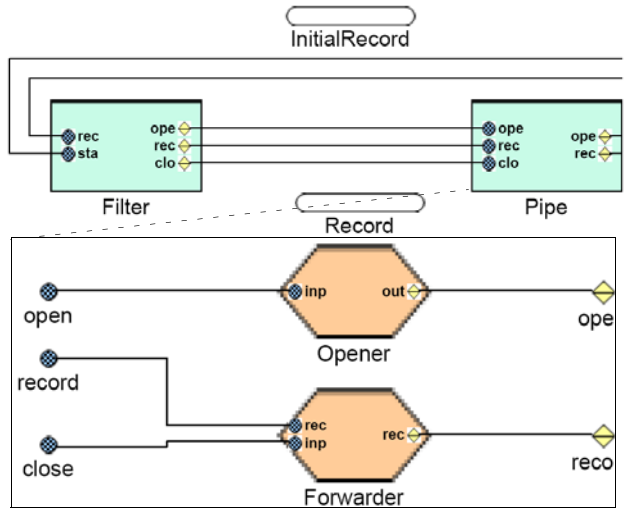
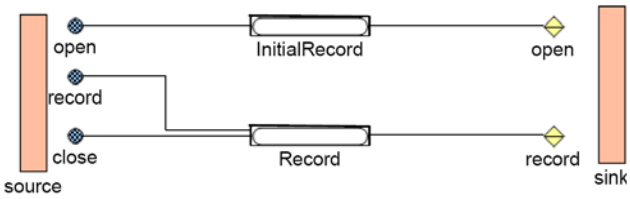


Figure 3. Architecture of photograph album software



a) Composition in pipeline style



b) Interfaces of Pipe in pipeline style

Figure 4. Pipeline style Alfa model

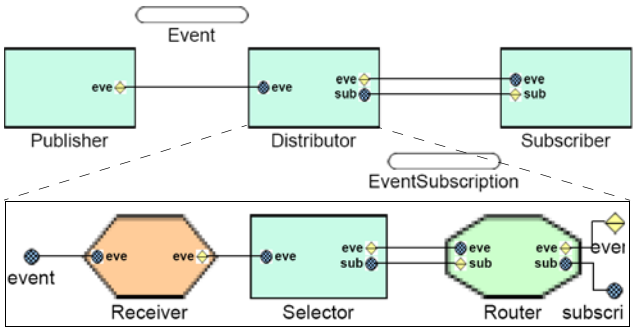


Figure 5. Event-based integration style Alfa model

example, the *Encoder* component only requires a *read* interface, but instantiating *Filter* would give it both a *read*

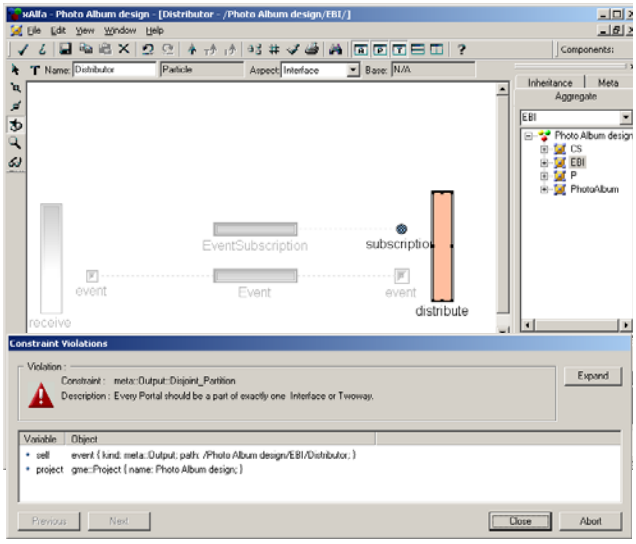
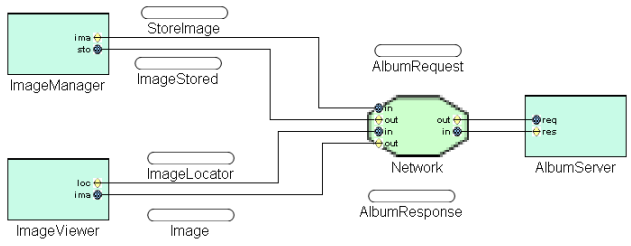
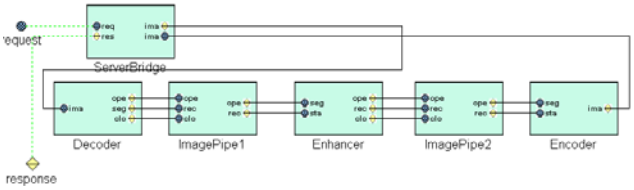


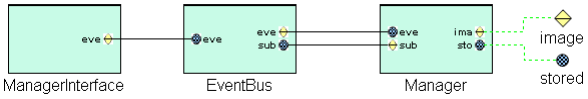
Figure 6. Example constraint violation in ViSAC



a) Top-level client-server style composition



b) Album Server composition in PF style



c) Image Manager composition in EBI style

Figure 7. Photo album architectural composition

and a *write* interface. If on the other hand, an *Image Viewer* is interested in communicating with two *Album Servers*, it would not be possible to add another *request* interface and associated portal instances. As a result, the refinement of constituent templates in ViSAC involves creating 1) new constituents, 2) instances of portal templates being refined, and 3) new INTERFACEs or TWOWAYs to partition the portals created in step 2. The templates of constituents, INTERFACEs, and TWOWAYs are then inferred from the types of portals created in step 2 above.

5. Discussion

ViSAC supports the visualization of styles and architectures. We have used ViSAC for modeling twenty different styles for network-based systems and architectures using them. We find that model hierarchies in GME are highly suited for hierarchical composition of style-based architectures. ViSAC's support for systematically combining multiple user-defined styles in architectures is a significant improvement over existing techniques. Moreover, the xAlfa meta model also allows the construction of style-less architectures by avoiding the use of templates and instantiation completely. Instead the constituents and portals

Visual modeling techniques and rich user interface features, such as drag-and-drop creation of references, sub types and instances, of GME make architectural composition using ViSAC truly visual and simplify manual manipulation of styles and architectures. Further, the use of aspects in GME allows the separation of concerns when modeling styles and architectures. ViSAC uses three different aspects 1) for composition of constituents, 2) for interface specification, and 3) for specifying constraints and behavior.

GME's sub-typing techniques are useful for extending styles or combining existing styles into new hybrid styles, but inappropriate for refining styles into architectures. ViSAC allows sub-types of constituents and datums from different styles to be easily combined into hybrid styles. In fact, we have studied the relationships among twenty different styles for network-based systems [6] in the Alfa project using such sub-typing techniques. Such studies have helped us to better understand these styles, their similarities and their differences.

An important benefit of using GME for modeling compositions of architectural primitives is its support for recording rich semantics of visual languages. The result is that well-formedness of style and architectural compositions can be automatically analyzed by ViSAC. The OCL-based notation used in GME for specifying dynamic semantics closely matches set- and relation-based composition theory of Alfa. In GME, constraint checking can be triggered off by a variety of user events. By correctly choosing events that trigger the checking of a GME constraint, ViSAC interactively helps the user prevent errors. More importantly, by breaking down the task of detecting malformation into smaller tasks, each performed at different levels of hierarchy, this approach reduces the overall complexity of validating style and architectural compositions.

Defining the semantics of xAlfa using MCL, in most part proved to be an easy and intuitive exercise, mainly due to the expressiveness of MCL. But we have recognized a few shortcomings of GME in this regard. Firstly, abstract kinds (e.g., Constituent in xAlfa) cannot be used in MCL expressions. This leads to much repetition of constraint expressions, which prove to be difficult to maintain. Also, GME informs users of constraint violation through a pop-

up dialog rather than by visually highlighting the location of such errors. This has frequently made it difficult to comprehend error messages.

Style and architectural models can be composed only partially using visual techniques. An important component of style models are expressions, which are usually recorded as ASCII text. However, certain stylistic constraints, such as on behavior, and architectural behavior are automata expressions, which could be captured as visual models themselves. We currently record all such expressions as text, leaving room for improvement in the future.

We are currently in the process of defining an interpreter for xAlfa, which would be used to analyze style conformance of architectures as well as to support code synthesis. A portion of the style conformance checking can be performed directly as MCL constraints on datums, ducts, portals, interfaces, and constituents. We are exploring this direction as it would integrate a part of style conformance checking into the ViSAC infrastructure and provide a uniform interface for reporting errors to a user.

An important characteristic of the meta modeling process is iterative development. As a result, a few dozen versions of the xAlfa paradigm were developed over the course of the last four months, and we have experimented with the resulting versions of ViSAC. Whenever xAlfa versions differ in minor changes, ViSAC can upgrade existing models to newer versions automatically upon user request. Even when upgrades fail due to significant differences between xAlfa versions, it is possible to work with existing style and architectural models using their outdated paradigms. The support for model upgrades in GME has proven absolutely essential for rapidly developing ViSAC.

The free availability of GME has allowed us to rapidly create a DSME for Alfa. Naturally, certain restrictions of GME have repercussions on its use for architectural modeling. The most important such restriction is that on model sub-typing and instantiation. Such restrictions lead to manual and tedious steps for refinement of architectural elements from style templates. Using GME infrastructure, custom components that manipulate GME models can be integrated to help reduce the effort involved in refining style templates. One such technique that we would like to explore is the use of wizard add-ons for specific tasks such as to refine a constituent instance from a constituent template.

6. Related Work

It has long been known that pictorial representations greatly aid cognition [10]. Thus, it comes as no surprise that much research effort has been devoted to the study of visual languages (VL). VL is a term generally used to refer to *a language with alphabet consisting of visual representations that are used for human-human and human-computer interactions* [20]. VL research deals with various topics that range from model analysis of its representation's spatial and topological properties to issues related with their efficient definition. VLs are commonly defined in terms of their syntax and semantics [20].

Recently much interest has developed in the area of *domain-specific visual languages* (DSVL) [26], from which our work draws greatly. These approaches rely on the definition of VLs that correlate closely to the problem domain. Based on these VLs, domain models are built which in turn can be transformed to other notations, including code. Tools such as GME [11], MetaEdit [18] and Dome [4] employ this approach. These tools use DVSLs to automatically generate *domain-specific modeling environments* (DSMEs). These environments provide the capabilities to define a DSVL, to generate environments for building domain-specific models in a given DSVL, and mechanisms to manipulate these models. We have chosen to use GME as our tool of choice over the other mentioned tools due to its unique combination of free availability, their close relationship to standard notations (UML and OCL), and clear extension mechanisms.

Our work parallels UML 2.0 and its related standards in many ways. The four layer meta-modeling approach followed by GME is the same used by OMG to define UML [22]. MCL, the language used to define the semantics of our notation, is compliant with OCL 1.4 [21]. UML 2.0, as a part of its *Superstructure* specification [22], defines the syntax, semantics and the visual notation for representing software architectures (in terms of components, connectors, ports, interfaces and collaborations). Alfa does all of the above while also providing a framework for defining architectural styles. Further, Alfa's composition theory is aimed at checking style conformance of the composed architectures. While UML aspires to be a "Universal" modeling language [5], the focus of Alfa is strictly on modeling styles and architectures. ViSAC could have been implemented as an extension to UML 2.0, but the lack of easily available techniques for customizing UML rule out any such possibility. However, when such techniques do become available, we will take advantage of them to construct a tool for Alfa.

Finally our work is closely related to the research in software architectural modeling environments. Most such tools tend to be style and notation specific. DRADEL for the C2 architectural style [15] and Weaves and its related toolset [8] are a few examples. ViSAC is style-neutral and aids an architect in defining architectural styles as well as architectures based on these styles. Generic tools such as ArchStudio[1] also support style-neutral architectural description, but do not allow the definition of styles. We also found the tree based user interface of ArchStudio to be cumbersome for hierarchical architectural compositions. The work closest to our own, is AcmeStudio [25], which also allows the definition of styles and multi-style architectures. However, AcmeStudio is not based on any notion of architectural primitives, and does not support behavioral and data type information in architectures. Moreover, it does not support the use of multiple styles at the same level of architectural hierarchy or the use of style-less architectural elements, both supported by ViSAC.

7. Conclusions

This paper has discussed the rapid development of a visual tool for architectural modeling using a domain-specific modeling environment, ViSAC. The main objective of ViSAC is to facilitate the interactive design of style-based software architectures. ViSAC has been implemented by configuring GME through a meta modeling process. Alfa's composition theory matches the concepts underlying GME: model hierarchy, atoms, connections, sets, references, multiple aspects, and constraints, thus simplifying the design and implementation of ViSAC. However, lack of flexible semantics for sub-typing in GME make style refinement slightly tedious.

Architectural styles and software architectures are both modeled is ViSAC on the basis of a GME paradigm called xAlfa. xAlfa reifies Alfa's composition theory by defining the syntax and semantics of a visual language for composing styles and architectures from Alfa's primitives. xAlfa's syntax enables the hierarchical composition of software architectures based on possibly multiple, user-defined styles. In addition to preventing malformed compositions, ViSAC can be easily extended to support the style conformance of architectures according to their style(s). Moreover, we are integrating ViSAC with a compiler for xAlfa architectural models, called alfaac, to check stylistic constraint expressions defined by a style designer. Another important objective of alfaac is to generate source code for architectural compositions.

ViSAC has been used for modeling twenty different styles for network-based systems. This paper partially illustrated the definition of three different styles and their use in modeling architectures. ViSAC provides interactive support for designing styles and architectures by alerting the user to their malformation. However, the design of style-based architectures tends to be tedious as style templates do not lend themselves easily to GME's sub-typing rules. We are currently engaged in tiding over such differences by designing task-driven wizards that leverage support for customization in GME.

8. References

[1] ArchStudio, http://www.isr.uci.edu/projects/archstudio

[2] Chen, M., Tang, M., and Wang, W. Software Architecture Analysis - A Case Study. *Proc. COMPSAC '99*, Phoenix, AZ, October 1999.

[3] Dashofy, E., van der Hoek, A., and Taylor, R. N. An Infrastructure for the Rapid Development of XML-Based Architectural Description Languages. *Proc. ICSE-24*, May 2002.

[4] Dome, http://www.htc.honeywell.com/dome

[5] Engels, G., Heckel, R., and Sauer, S. UML - A Universal Modeling Language? In M. Nielsen, D. Simpson (eds.), *Proc. ICATPN-2*, Aarhus, Denmark, LNCS vol. 1825, Springer-Verlag, 2000.

[6] Fielding, R. Architectural Styles and the Design of Network-Based Software Architectures. Ph. D. Dissertation, University of California at Irvine, 2000.

[7] GME 3 User's Manual, Version 3.0, Vanderbilt University, March 2003.

[8] Gorlick, M. M. and Razouk, R. R. Using Weaves for Software Construction and Analysis. *Proc. ICSE-13,* Austin, Texas, USA, 1991.

[9] Jackson, D. Automatic Analysis of Architectural Style. Unpublished Manuscript, MIT Laboratory for Computer Sciences, Software Design Group.

[10] Kulpa, Z. Diagrammatic representation and reasoning. *Machine Graphics & Vision*, 3: 77--103, 1994.

[11] Ledeczi, A. et al. The Generic Modeling Environment. *Proc. WISP '01*, Budapest, Hungary, May 2001.

[12] Ledeczi A., et al. Composing Domain-Specific Design Environments, *IEEE Computer*, November, 2001.

[13] Matlab/Simulink, http://www.mathworks.com.

[14] Medvidovic, N., Mikic-Rakic, M., Mehta, N. R., and Malek, S. Software Architectural Support for Handheld Computing. *IEEE Computer Special Issue on Handheld Computing*, September 2003.

[15] Medvidovic, N., Rosenblum, D., and Taylor, R. N. A Language and Environment for Architecture-Based Software Development and Evolution. *Proc. ICSE-21*, Los Angeles, California, USA, May 1999.

[16] Mehta, N. R. and Medvidovic, N. Composing architectural styles from architectural primitives. *Proc. ESEC-10/FSE-11*, Helsinki, Finland, September 2003.

[17] Mehta, N. R. and Medividovic, N. Composition of Style-Based Software Architectures from Architectural Primitives. Technical Report USC-CSE-04-503, University of Southern California, 2004.

[18] MetaEdit, http://www.metacase.com

[19] Monroe, R. T. and Garlan, D. Style-Based Reuse for Software Architectures. *Proc. ICSR-4*, Orlando, Florida, USA, April 1996.

[20] Narayanan, N. H. and Hbscher, R. Visual language theory: Towards a Human computer interaction perspective. In Marriott, K. and Meyer, B. (eds), *Visual Language Theory*, Springer, New York, 1998.

[21] Object Management Group, Unified Modeling Language Specification, Version 1.4. http://www.omg.org/cgi-bin/apps/doc?formal/01-09-77.pdf, September 2001.

[22] Object Management Group, UML 2.0 Superstructure Specification. http://www.omg.org/cgi-bin/apps/doc?ptc/03-08-02.pdf, August 2003.

[23] Object Management Group, UML 2.0 Infrastructure Specification. http://www.omg.org/cgi-bin/apps/doc?ptc/03-09-15.pdf, December 2003.

[24] Perry, D. E. and Wolf, A. L. Foundations for the Study of Software Architectures. *ACM SIGSOFT Software Engineering Notes*, 17, 40-52, 1992.

[25] Schemrl, B. and Garlan D. AcmeStudio: Supporting Style-Centered Architecture Development. Unpublished Manuscript, CMU School of Computer Science.

[26] Tolvanen, P, et al.First Workshop on Domain Specific Languages, October 2001.

[27] Williams, L. G., and Smith, C. U. PASA^{SM}: A Method for the Performance Assessment of Software Architectures. *Proc. IWSP*, Rome, Italy, 2002.