A model-driven approach for managing software architectures with multiple evolving concerns

Ivano Malavolta
Dipartimento di Informatica, Università dell'Aquila, L'Aquila - Italy
ivano.malavolta@univaq.it

ABSTRACT

Architecture Description Languages (ADLs) are the means to define the software architecture of a complex system. ADLs are strongly related to stakeholder concerns: they must capture all design decisions fundamental for system’s stakeholders. Stakeholder concerns are various and ever evolving, thus it is impossible to capture all system’s concerns with a single ADL. The evolution of a concern may cause the used ADLs (or even the whole architecture framework) not to fit any more with the system-of-interest.

In this work we propose an approach for describing software architectures with multiple and evolving stakeholder concerns. Under this perspective, the proposed approach allows (i) to define customized ADLs depending on the concerns held by the system’s stakeholders, (ii) to provide interoperability between either customized and already existing ADLs, (iii) to realize architecture frameworks in which viewpoints, views and languages are set up depending on the system’s stakeholders concerns. The approach is based on model-driven engineering technologies.

Categories and Subject Descriptors
D.2.11 [Software Engineering]: Software Architectures—Domain-specific architectures; D.2.11 [Software Engineering]: Software Architectures; D.2.10 [Software Engineering]: Design

General Terms
Design, Modeling.

Keywords
Software Architecture, ADL, architecture framework, modeling.

1. INTRODUCTION

During the development of a complex system, it is fundamental to correctly and clearly specify its software architecture (SA) [13]. Nowadays the software architecture of a system is typically expressed in terms of multiple views, each view addressing a given set of system concerns which are held by various stakeholders. A view expresses the architecture of the system-of-interest in accordance with a specific architecture viewpoint. It establishes the conventions for constructing, interpreting and analysing an architecture view. Typically, a specific view (or a set of views) of the system are defined by means of an Architecture Description Language (ADLs) [11]. Fundamentally, an ADL is a kind of domain specific language for specifying software architectures. While developing a complex system, ADLs, viewpoints, views and notations are usually coordinated in the context of an architecture framework [8].

According to what said so far, there are some issues affecting current architecture frameworks and ADLs. Current ADLs are still not aligned with the evolved concept of software architecture: emerging requirements like language extensibility, customization, or multiple views management are still not adequately supported. This is because the concept of software architecture is getting broader and takes into consideration also concepts like stakeholders concerns and design decisions [8, 2, 13] which are not adequately captured in current ADLs. Moreover, since it is impossible to capture all such concerns with a single, narrowly focused notation [10], we must accept the multiple ADLs for modelling software architectures, therefore considering each ADL aimed at solving specific stakeholder concerns. This calls for ADLs interoperability. Furthermore, architecture frameworks tend to be closed: constituent elements are defined in different non-standard ways, they are re-usable, and the customization of a framework (typically after an evolution of the stakeholders concerns) requires a complete rework. Another interesting issue is that ADLs and architecture frameworks must react to the possible evolution of system’s stakeholder concerns (e.g., new concerns may arise, or existing concerns may change).

The main contribution of this thesis is to overcome the above mentioned issues by developing an approach for describing SAs (via different ADLs) which treat multiple and evolving stakeholder concerns as first-class elements. The proposed approach allows to: (i) define customized ADLs depending on the concerns held by the system’s stakeholders; (ii) provide interoperability between either customized and already existing ADLs; (iii) realize architecture frameworks in which viewpoints, views and languages are set up depending on the system’s stakeholders concerns.

In the remainder of this paper, Section 2 presents the research questions, Section 3 and Section 4 present the approach in terms of both the results reached so far and the expected ones. Section 5 describes the plan for the evaluation of the results. Finally, Section 6 presents some related work and Section 7 concludes the paper.
2. RESEARCH QUESTIONS

Architecture description languages are strongly related to system’s stakeholder concerns, thus an ADL must provide a good level of abstraction and the right constructs for addressing them. Clearly, the quality of the resulting architecture descriptions heavily depend on how the stakeholder concerns of the current system are supported. This issue depends on (i) the concepts supported by the ADLs and (ii) the architecture framework managing architecture languages, models, design decisions and the concerns themselves.

Goal of this thesis is to answer the following research questions:

RQ1. Focusing on a single language, is it feasible to customize or adapt an ADL in order to get it closer to either the domain, development process or stakeholder concerns of the system?

RQ2. A set of ADLs is typically used for describing the software architecture of a complex system. Is there a way to automatically keep aligned models conforming to different ADLs? Is it possible to reuse their commonalities and let them interoperate? Further, is there a scalable way to add (or remove) a notation from that set of ADLs?

RQ3. Is it possible to reuse already existing viewpoints, languages, design decisions in order to ‘shape’ an architecture framework to current system’s stakeholder concerns?

RQ4. Is it possible to provide a generic unified approach for specifying SAs that treats (multiple and evolving) stakeholder concerns as first-class elements and semi-automatically adapts to them?

Stakeholder concerns may also evolve while architecting; this is an orthogonal issue common to all our research questions. The evolution of a concern may affect an architecture description at different levels: it may affect the constructs of a single ADL, a specific ADL may become unnecessary, it may call for the use of additional ADLs, it may affect also the whole architecture framework. We consider also this issue while addressing the above mentioned research questions.

This thesis aims to investigate on how current MDE technologies helps in answering our research questions with a focus on reuse and automation.

3. CURRENT RESULTS

This section presents results obtained so far concerning the research described in this paper. From an abstract point of view, we had interesting preliminary results in trying to answer the first three research questions described in Section 2, whereas we still have to elaborate on RQ4. More specifically, the efforts in answering RQ1, RQ2 and RQ3 converged in the development of three different approaches called byADL, DUALLy and MEGAF, respectively. Next sections give a brief overview of the results about each of them. Understanding how the three approaches function together in a unified environment is still an ongoing work (see Section 4).

3.1 Building ADLs with byADL

BYADL [3] is an approach for developing next generation ADLs according to their recently evolved requirements (e.g., directly support domain-specific concerns, analysis features, tool support). Figure 1 presents an high-level overview on the functioning of BYADL. BYADL provides an incremental approach to build customized and customizable ADLs starting from an already existing one (ADLx in the figure); the new ADL is build by means of semantic extension mechanisms. The main input of BYADL is represented by the metamodel of the ADL to be extended; it is contained into a repository of metamodels, together with other metamodels representing either domain, analysis constructs, views, methodologies, and so on. The metamodel of ADLx is extended by applying specific metamodel composition operators. Conceptually, the composition engine allows to extend the ADL with domain-specific concerns, with new architectural views, with analysis constructs or with methodology and process concepts (see lower part in Figure 1), depending on the system’s stakeholder concerns. Moreover, since the byADL approach is incremental, its composition mechanism can be used to easily and quickly update an already existing ADL after a system concern is evolved. The ADL could be also simply customized; this is done by composing it with ad-hoc metamodels (see left upper part in Figure 1) representing only the customization aspects of the language (i.e., only the entities that change or must be added).

The ADL obtained at the end of the process consists of (i) abstract syntax, i.e., the metamodel obtained by means of the composition mechanisms, (ii) a set of concrete syntaxes, i.e., automatically generated textual and graphical notations to visualize and edit models, and (iii) semantics describing the meaning of each language construct.

There are three different possibilities to produce an editor for the ADL being developed in BYADL: tree-based, textual, and graphical. Each editor is characterized by different levels of usability and requires different effort for the customization (if needed).

The semantics of the extended ADL is given by means of semantic relationships between the language’s elements and elements of a core set of architectural concepts called A₀ (see Section 3.2). By means of such relationships, the elements of the ADL implicitly inherit the semantics of A₀.

In BYADL specific model transformations are able to reflect the models defined within the newly created ADL, back to the original tools. These transformations are automatically generated by higher-order transformations and are fundamental for analysing architecture models using their original tools.

For the details on BYADL, please refer to [3] or its website 1.

3.2 Providing interoperability with DUALLy

DUALLy [10] is an approach to create interoperability among architectural notations. They may be either in the form of UML profiles or “canonical” ADLs. DUALLy provides a mechanism to automatically generate transformations allowing to pass from an architectural notation to another and vice versa. The configuration of DUALLy is depicted in Figure 2.

1BYADL website: http://byadl.di.univaq.it
Conceptually, **dually** is a transformation framework that permits to transform an architecture model \( m_1 \) into a corresponding architecture model \( m_2 \). Each \( m_1 \) conforms to its ADL, that is either a metamodel or a UML profile.

**Figure 2: Overview of Dually**

Such a transformation passes through \( A_0 \): a pivot metamodel specific for the SA domain. The main purpose of \( A_0 \) is to provide a centralized set of semantic elements with respect to which relations must be defined. As shown in Figure 2, for the realization of **dually** we chose a star architecture in which \( A_0 \) is the centre. Whenever a model \( m_1 \) has to be transformed into \( m_2 \), a two-step process is performed by transforming \( m_1 \) into \( m(A_0) \), and successively \( m(A_0) \) into \( m_2 \). \( A_0 \) has been defined as general as possible to ensure that **dually** is able to potentially represent and support any kind of architectural representation. Thanks to the \( A_0 \)-centred star architecture a linear relationship between the selected language and \( A_0 \) is created, thus reducing the number of transformations needed. Also, adding an ADL from the topology of notations is straightforward: the software architect must simply link the new ADL to \( A_0 \), without considering all the other notations in the topology. The same rationale holds for removing an ADL from the topology of notations. However, while the star architecture reduces the number of needed transformations, a possible drawback could be that two different ADLs share concepts that are not contemplated in \( A_0 \), thus the quality of the transformation is lowered down only because those common concepts are not part of \( A_0 \). We solved this issue by designing \( A_0 \) as extensible, so that suitable extensions may be defined in order to consider additional elements in \( A_0 \).

**dually** operates at both the metamodeling and modeling levels of abstraction. This provides a clear separation between MDE experts and software architects. At the metamodeling level, MDE engineers provide a specification of the architectural language (e.g., ADL) in terms of its metamodel or UML profile. They then define a set of links so as to semantically relate architectural concepts of the language with the equivalent elements in \( A_0 \). At the modeling level, software architects specify the SA using their preferred ADL or UML-based notation. Starting from the semantic links at the metamodeling level, **dually** allows the automatic generation of model-to-model transformations which allow to automatically translate \( m_1 \) (written according to ADL) into a corresponding model (conforming to \( A_0 \)) and vice versa. Now the \( A_0 \) model may be transformed towards any other notation in the **dually** topology. The model transformation engine is completely hidden to a software architect, that can continue using familiar architectural notations and tools.

This thesis contributes to **dually** in (i) defining its mechanisms using recent model-driven technologies (like model weaving, model-to-model transformations, tracing mechanisms), the definition of the \( A_0 \) metamodel, and (iii) techniques for extending \( A_0 \). More details on **dually** can be found in [10] or its website.

### 3.3 Realizing architecture frameworks with MEGAF

**Megaf** [7] is an infrastructure for realizing architecture frameworks with a focus on (i) reusing already defined viewpoints and languages and (ii) considering stakeholders and their concerns as part of the framework itself. **Megaf** is built on the conceptual foundations provided in the ISO/IEC 42010 standard. **Megaf** allows software architects to create an architecture framework; it also allows to specify the necessary views, viewpoints, stakeholders, concerns of the system and navigate among them.

**Figure 3: Overview of Megaf**

**Megaf** can be considered as an extensible repository of viewpoints, views, model kinds, architecture models, system concerns, and stakeholders. Correspondences and correspondence rules between arbitrary elements, including stakeholders, system concerns, models, etc. can be created in **Megaf**. More precisely, they enable the architect to express and enforce relations both between various elements inside an architecture description and across architecture descriptions (such as for product lines or systems of systems).

Figure 3 gives an idea on how **Megaf** supports the creation of an architecture framework depending on the various stakeholder concerns of the system. **Megaf** provides functionalities that allow software architects to create their own framework by properly selecting among artefacts previously defined and resident inside **Megaf**. Once the framework has been defined, it can be used to realize the architecture description of the system-of-interest. In other words, **Megaf** allows to create an architecture description conforming to the architecture framework, to specify views that are governed by viewpoints defined in the architecture framework and that address some previously-specified system concerns.

Under this setting, the evolution of a system concern simply triggers a refactoring of the **Megaf** underlying infrastructure, that is actually formalized as a specific kind of model called megamodel. It also allows to check consistency among arbitrary architectural elements. **Megaf** also supports different kinds of checks:

- conformance of an architecture description with ISO/IEC42010;
- compliance of an SA description with an architecture framework;
- satisfaction of framework-specific rules;
- satisfaction of rules depending only on the system-of-interest.

The use of megamodeling promotes the reuse of any architectural element that resides in the **Megaf** infrastructure. Intuitively, in **Megaf** a framework can be created either by linking and reusing existing elements or adding new ones if needed. For more information about **Megaf**, please refer to [7] or to its website.

---

2 **dually** website: [http://dually.di.univaq.it](http://dually.di.univaq.it)
3 **Megaf** website: [http://megaf.di.univaq.it](http://megaf.di.univaq.it)
4. EXPECTED RESULTS

This section presents the future results we expect from this thesis work. Each expected result is defined in the context of its corresponding research question.

RQ1. It is planned to investigate on extending BYADL towards a generic means to merge, split, or customize views and viewpoints in an integrated MDE environment. Moreover, the BYADL migrating engine (e.g., the transformations towards original notations) will be enhanced so that it may support regression analysis. This enables software architects to understand if results of performed analysis remain valid even after an SA model is changed. We will investigate the feasibility of using BYADL to create a new ADL from scratch by considering a generic metamodel for SAs as a starting notation. The main question here is on what should be the starting point of the composition process.

RQ2. Regarding interoperability among ADLs, we are investigating on how to fully automatize it. That is, currently in DUALLY a set of semantic links must be established between each ADL and the \(A_0\) pivot metamodel. We are investigating on how to automatically generate that set of semantic links; this issue may be tackled by considering some heuristics or ontology-based approaches. Another important aspect is that of combining the DUALLY infrastructure with synchronization and change propagation approaches, so that only model changes are “exchanged” among the various ADLs. As future work it is also planned to enhance the management of the \(A_0\) pivot metamodel. In fact, it is under study how to promote \(A_0\) from a single pivot metamodel into a “cloud” of pivot metamodels together with a mechanism to manage such a cloud (e.g., the mechanism manages the entry/exit points of the cloud and how models travel within the cloud itself). Technically, this may be achieved by allowing to extend \(A_0\) through a set of well-formalized operators and providing specific mechanisms for applying and managing those operators. By doing this, ADL-to-ADL transformations will be much more accurate. A more generic aspect regards techniques to check the quality of the transformations between ADLs. This issue may be studied from both a semantic and syntactic point of view. Addressing this issue at the semantics level will likely provide better results about the quality of the transformations. Concepts like semantic anchoring, semantic equivalence between models in MDE, model transformation testing will be considered in this part of research.

RQ3. An interesting direction for future investigation is about understanding how concepts like architectural knowledge, design decisions, or modeling conventions may be associated with views and viewpoints within an architecture framework. Furthermore, it is also evident that the more MEGAF is populated with architectural elements, the more effective becomes their reuse. As the MEGAF repository grows larger, advanced features for searching and querying elements in the repository will be developed. At the moment in MEGAF an artefact can be reused as it is. Modifications can be made by hand, possibly starting from a copy of the existing element. We are currently investigating on the feasibility to reuse the BYADL composition engine in order to provide a more formalized (and thus with room for automation) way for extending and customizing already defined architectural entities, like frameworks, languages, models, viewpoints, correspondences.

RQ4. As previously stated, one of the main expected results is that of addressing RQ4. That is, to provide a generic approach supporting all the mechanisms presented in the previous sections in a seamless, unified way. Modern software architects may utilize an implementation of the proposed approach as the basis for developing either ADLs definitions, SAs descriptions and perform different kinds of analysis on them. Moreover, an implementation of the proposed approach should provide also facilities for managing the description of the system in terms of multiple views and some mechanism to check (and optionally enforce) the synchronization among them. Such a unified approach will be used by stakeholders operating in all the life-cycle activities of the system (like requirements, development, testing, maintenance) and will act as a central reference point for the correct development of the system. This implies that the implemented approach should provide different solutions and system-views, depending on the current stakeholder using it. It should also provide different views of the architecture depending on the current stakeholder and filter system information. An implementation of the unified approach should provide only the information related to the concerns held by the current stakeholder, while hiding other aspects of the system depending on the stakeholder’s access rights; this results in a much more cognitively manageable and well organized architecture description of the system.

5. EVALUATION PLAN

As detailed in Section 3, a set of prototypes addressing RQ1, RQ2 and RQ3 has been implemented; they serve as an experimental test bed for our research.

We evaluated the BYADL prototype through a set of exemplary scenarios we considered representative enough for testing it. In the context of one of those exemplary scenarios, we re-used models derived from a naval industrial case study (it will be presented later in this section). However, we still did not fully validate BYADL via real industrial case studies; this is part of future work. It is also planned another experiment on BYADL: it consists in a retrospective study on the Koala ADL [14]. More specifically, it is well known that the Koala ADL has been defined by considering the Darwin language as starting point and by extending it with concepts from the consumer-electronics domain. In this study the efforts of producing the Koala language manually and via the BYADL approach will be compared; this will provide some quantitative insight on the effectiveness of BYADL.

DUALLY has been validated with two main case studies:

- in [9] we considered a Flight Control System (FCS) modelled inAADL [4] and then we used DUALLY to automatically transform FCS models towards other architectural notations;
- in [10] we considered the software architecture of a real system called Integrated Environment for Communication on Ship (IECS); this was an industrial case study coming from a collaboration with a company operating in the naval communication domain.

MEGAf has been validated via an industrial web-based health information system case study. The case study is based on the HEALTH-WATCHING system and forms the basis for a full example of applying the ISO/IEC 42010 standard for SA descriptions. Since the above mentioned approaches are featured with an already implemented prototype, it has been possible to test them on real case studies; validating them with further industrial case studies is part of future work. However, since constraints and concerns related to the unified approach corresponding to RQ4 are still under study, a prototype implementing that approach is still missing. Realizing such a prototype will enable to validate the whole approach with real industrial case studies. This may provide interesting and unforeseen insights on the overall approach proposed in the thesis.

6. RELATED WORK

Many works have been done in the direction of getting an ADL close to the domain of the system-of-interest and adapting it to an
initial set of concerns. They mainly focus on the definition of extensible and domain- and style-specific ADLs [4, 1, 5, 12].

xADL [1] as well as its core xArch, are based on XML and thus fully extensible. However, XML schemas do not provide facilities to define the semantics of individual elements but only the syntax. The semantics is fundamental in order to define a common and well defined means to interpret the syntactic concepts. The BYUADL approach provides a semantic extension mechanism. Architecture frameworks for ADLs generated by MEGAF support all the features of the xADL tooling set, including both textual and graphical layout, Java data binding libraries, automated analysis (thanks to DUALLY) and so on.

UML [12] has been used for modeling SAs and different profiles have been proposed. While extension mechanisms of UML allow software architects to define stereotypes and tagged values (which can be used to better capture domain-specific aspects) these extensions cannot fully represent all stakeholder concerns.

AADL [4] was designed as an extensible language supporting modelling with multiple aspects and viewpoints. The extension mechanisms of AADL include the definition of custom properties to specify additional ADL-specific analysis and/or generic information to be attached on the architectural design. Unfortunately, AADL does not provide automatic support for its extensibility (like the composition operators and the migrating engine in BYUADL).

All these attempts to create extensible and domain- and style-specific ADLs do not sufficiently take into account the problem of reusing already defined extensions. Extensions should be defined as ADL-independent and collected in libraries.

Acme [3] is one of the very first technologies to tackle the problem of interchange in ADLs, and so it is strongly related to the ADLs interoperability issue. Acme was born as a simple, multi-style ADL framework also providing foundations and mechanisms to extend itself. More precisely, Acme provides tooling extension points allowing other tools to physically read and write Acme descriptions. Properties can decorate each element; they are uninterpreted and it is up to user-written tools to properly parse and use them. BYUADL provides mature extension mechanisms enabled also by the current MDE techniques, and DUALLY provides a semantic-driven approach for models interchange.

The role of $A_0$ in DUALLY is similar to the Klaper[6] language in the field of performability. Indeed Klaper has been proposed as a pivot modelling language within a star topology. However in the Klaper-based methodology model transformations are not horizontal and model transformations are not derived from semantic bindings: they must be manually developed.

Since their initial definition, many architecture frameworks have been proposed, e.g., RM-ODP, GÉRAM, TOGAF, and DODAF. The recurring idea among these is that an architecture framework is a prefabricated structure that one can use to organize an architecture description into complementary views. Most architecture frameworks in use are closed (i.e. they cannot be easily adapted to new needs/concerns), and the construction of existing ones requires a complete rework since reuse is not supported. Automated support (where it exists) follows the predefined viewpoints and models; support for developing extension of architecture frameworks in terms of new viewpoints does not exist.

7. CONCLUSION

This paper presented an approach and current results towards the definition and design of a model-driven infrastructure for managing software architectures with multiple evolving concerns. Currently, research prototypes related to some aspects of the proposed approach have been already implemented (see Section 3). However, a unified and fully integrated environment in which all the parts of the proposed approach (and their corresponding prototypes) coexist is still under study.

As next steps towards the dissertation, I will focus on three main directions: (i) elaborate further and refine the approaches and prototypes related to $RQ1, RQ2$ and $RQ3$; (ii) investigate on a unified approach comprising them, thus addressing $RQ4$; (iii) continuously study MDE technologies in order to deal with the two previous points with more powerful techniques and elegant solutions.

Acknowledgments

This work is partly supported by the Italian PRIN d-ASAP and FIRB ArtDeco projects.

8. REFERENCES


