Providing Architectural Languages and Tools Interoperability through Model Transformation Technologies

Ivano Malavolta, Henry Muccini, Patrizio Pelliccione, Damien A. Tamburri

Technical Report TRCS 004/2008

The Technical Reports of the Dipartimento di Informatica at the University of L'Aquila are available online on the portal http://www.di.univaq.it. Authors are reachable via email and all the addresses can be found on the same site.

Dipartimento di Informatica
Università degli Studi dell'Aquila
Via Vetoio Loc. Coppito
I-67010 L'Aquila, Italy

http://www.di.univaq.it

Technical Report TRCS Series
Providing Architectural Languages and Tools
Interoperability through Model Transformation
Technologies

Ivano Malavolta, Henry Muccini,
Patrizio Pelliccione, Damien A. Tamburri
Dipartimento di Informatica, Università dell’Aquila, Via Vetoio, 67100 L’Aquila,
{ivano.malavolta,muccini,pellicci,damien.tamburri}@di.univaq.it

Abstract

Many architectural languages have been proposed in the last fifteen years, each one with the chief aim of becoming the ideal language for specifying software architectures. What is evident nowadays, instead, is that architectural languages are defined by stakeholder concerns. Capturing all such concerns within a single, narrowly focused notation is impossible. At the same time it is also impractical to define and use a “universal” notation, such as UML. As a result, many domain specific notations for architectural modeling have been proposed, each one focussing on a specific application domain, analysis type, or modeling environment. As a drawback, a proliferation of languages exists, each one with its own specific notation, tools, and domain specificity. No effective interoperability is possible to date. Therefore, if a software architect has to model a concern not supported by his own language/tool, he has to manually transform (and eventually keep aligned) the available architectural specification into the required language/tool.

This paper presents DUALLY, an automated framework that allows architectural languages and tools interoperability. Given any number of architectural languages and tools, they can all interoperate thanks to automated model transformation techniques. DUALLY is implemented as an Eclipse plugin. Putting it in practice, we apply the DUALLY approach to ACME and DARWIN ADLs and then transform complex DARWIN specifications into ACME models. We apply also the approach to AADL, somewhat different than ACME and DARWIN, in order to discuss the quality of DUALLY.

Index Terms


I. INTRODUCTION

Up to the present day, many languages for specifying and analyzing software architectures have been proposed (e.g. [11], [38], [37]) and historically classified into two generations [35]. A “first generation” of Architecture Description Languages (ADLs) going from 1990 to 2000, had the main purpose to design an ideal ADL [27] whose
chief aim was to enable support of components and connectors specification and their overall interconnection [39], [27], as well as composition, abstraction, reusability, configuration, heterogeneity, and analysis [44]. Later on, during the “second generation”, going from 2000 up to today, new requirements emerged, and new ADLs have been proposed to deal with more specific features [8], [40], such as configuration management, distribution, and product line modeling support.

As a result, a proliferation of architectural languages can be noticed today, each differentiated by slightly different architectural conceptual elements, syntax or semantics, focussing on a different operational domain, or only suitable for different analysis techniques\(^1\). Even the adoption of UML for modeling architectures (e.g. [41], [36], [25], [34]) is biased by different concerns: a number of UML profiles and extensions have been proposed for modeling different concerns, thus increasing even more the proliferation of architectural languages. These extensions cannot fully represent all aspects/features of every ADL, neither there is a unique way to model a single ADL (as already claimed in [18], [25], [37]), testifying that it is also impractical to have a “universal” notation.

As noticed in [35], one of the main reasons for such a long list of architectural languages is related to stakeholder concerns: a notation has to adequately capture design decisions judged fundamental by the system’s stakeholders, thus the notion of software architecture has been expanded and notations as well as approaches for modeling software architectures have themselves continued to evolve. Stakeholder concerns are various, ever evolving, and adapting to new environment requirements; hence it is impossible to capture all such concerns with a single, narrowly focused notation. Instead of a unique language for specifying software architectures, we must therefore accept the existence of domain specific languages for SA modeling, therefore considering each ADL aimed at solving specific stakeholder concerns (e.g. architectural styles, real-time constraints, data-flow-architectures, code generation, etc.). In Section V, among the existing ADLs we will extract and analyze those ADLs whose goal we consider close to ours in terms of effort invested in extensibility or interoperability mechanisms.

Very limited interoperability possibilities among tools and notations exist. In fact, assuming an SA specified in, e.g. ACME, needs to be model checked, a different architectural specification needs to be produced, e.g. in Darwin, since ACMEStudio does not support model checking. As soon as the model-checking analysis in Darwin/LTSA requires a modification in the SA, consistency between the new Darwin specification and its ACME counterpart has to be manually maintained.

These considerations led us to propose DULLY, a framework to create interoperability among ADLs themselves as well as UML. As conceptually shown in Figure 1\(^2\), DULLY permits to transform concepts of an architectural model M1 into the semantically equivalent concepts in the architectural model M2. Each Mi conforms to its MMi meta-model or UML profile. Every MMi can be a meta-model of an architectural language. Therefore, DULLY works at two abstraction levels: meta-modeling (upper part of Figure 1), and modeling (lower part of Figure 1).

At the meta-modeling level, model driven engineers provide a specification of the architectural language in terms

\(^1\)In an ongoing study we are conducting, we counted more than 50 ADLs proposed from academia and industry.

\(^2\)This figure will be refined in the following of this paper.
of its meta-model or UML profile. They then define a meta-transformation so as to semantically relate architectural concepts in MM1 with the equivalent elements in MM2.

At the modeling level, software architects will specify the SA using their preferred ADL or UML-based notation. DUALLY allows the automatic generation of model-to-model transformations which enable the software architect to automatically translate the M1 specification (written according to MM1) into the corresponding M2 model (conforming to MM2) and vice versa.

DUALLY is implemented as an Eclipse plugin. MMi meta-models are expressed in Ecore. MMi profiles can be realized using any UML tool which exports in UML2, the EMF-based implementation of the UML 2.0 meta-model for the Eclipse platform. A0 is represented as a UML profile. Transformations among meta-models/profiles and model-to-model transformations are implemented in the context of the AMMA platform [14]. Model-to-model transformations are then automatically instantiated, thus providing the possibility to automatically reflect modifications made on a model designed with one extension to one or even all of the other extensions.

DUALLY’s main advantages can be summarized as follows:

- DUALLY works at two abstraction levels, providing a clear separation between the role of model driven experts (i.e. the technical stakeholder) and that of software architects (i.e. the final users). The model transformation engine is completely hidden to a software architect, making DUALLY extremely easy to use;

As it can be noticed in Figure 1, the meta-transformation (and its corresponding generated transformation) relates MM1 to MM2 (as well as M1 to M2) passing through what we refer to as A0. A0 represents a semantic core set of modeling elements, providing the infrastructure upon which to construct semantic relations among different ADLs. It acts as a bridge among the different architectural languages to be related together. The why and how of A0 will be discussed in Section II. The direct arrow between MM1 and MM2 signifies that it is also possible to define a direct meta-transformation between them if there is the need to relate elements existing in both the meta-models/profiles which are not contemplated in the A0 profile.

Fig. 1. DUALLY Conceptual View

March 31, 2008 DRAFT
• DUALLY enables the transformation among formal ADLs, UML model-based notations and vice versa;
• Software architects can continue using familiar architectural notations and tools even reusing legacy architectural models;
• DUALLY enables both languages and tools interoperability;
• The meta-transformations among two architectural notations are defined once, and reused for each possible model.

The rest of the paper is structured as follows: Section II presents DUALLY and its main principles. Section III presents the technology used to engineer DUALLY and discusses its implementation. Section IV shows a complex example of the integration of Darwin and ACME within DUALLY and our experience in translating a Darwin model into an ACME specification. Section V describes related work. Section VI evaluates DUALLY and provides consideration on its implementation and usage; in particular, we will discuss how we deal with the quality of our transformation network. Section VII concludes the paper and envisions future research directions. Finally, a list of acronyms is provided in Appendix.

II. DUALLY: THE FRAMEWORK IN CONCEPTS

Figure 2 is a high level conceptual view of DUALLY. This picture shows the main purpose of DUALLY, that of allowing different formal ADLs and/or UML-based languages and tools for software architecture modeling to interoperate. Deeper down to the conceptual details, we present Figure 1, which is clearly a full instance of two branches from Figure 2’s more general view.

A number of key elements can be distinguished in both figures: meta-models/profiles, meta-model links, \( A_0 \), architectural models and model transformations. The two main users intended for DUALLY are meta-modelers, who will act mainly on the first three elements and software architects, acting mainly on the last two elements. While being the basic reference for new projects, \( A_0 \) also becomes the staging point for the complete and consistent migration for architectural information across any number of description technologies already integrated within DUALLY in the form of meta-models or profiles, by means of the process we call DUALLYzation. As clearly
shown in Figure 2, for **DUALLY**’s realization we chose a “star” architecture. $A_0$ is placed in the center of the star, while **DUALLY**’s transformation engine is in charge of maintaining the transformation network.

The depicted star architecture and the devised branches will enable multiple technology intercommunication as well as total level compliance, i.e. what is in one level can be transformed onto each inhabitant of that level. In Section II-A we will provide detail on $A_0$, its structure and meaning as well as a thorough explanation of its role within our transformation engine. **DUALLY** transformation technologies will be explained in Section II-B.

A. **DUALLY** Core Set: $A_0$

**DUALLY** inherits from both xArch [7] and ACME [9] the idea of identifying a core set of architectural concepts, hereafter referred as $A_0$ (see Figure 1). The main purpose of $A_0$ is to provide a centralized set of semantic elements with respect to which relations must be defined. Considering that many languages need to be related together (see Figure 2), instead of creating a point-to-point relationship among all languages, a linear relationship among the selected language and $A_0$ is created, thus reducing the number of connections. What may indeed happen is that two ADLs, say $ADL_i$ and $ADL_j$, share some domain specific concepts and these are not contemplated in $A_0$. In this case, direct link among $ADL_i$ and $ADL_j$ can be defined. Note that, should this case rise, the connection between $ADL_i$ and $ADL_j$ through $A_0$ has to be forbidden (for the current status of **DUALLY**) to avoid two possibly inconsistent generated models, one being generated by the direct link and the other obtained through the default “$A_0$ bridge”. In the future work section we will provide some directions that will explore and overrun this limitation.

The selection of elements within $A_0$ was mainly guided by the principle of maintaining our base notation as general as possible to ensure that **DUALLY** is able to potentially represent and support any kind of architectural representation (i.e. formal ADLs or UML-based languages).

$A_0$ acts as a central pillar of the model transformation network. It is the base language that every technology may use to keep “aligned” with any other integrated one. All the passages between technologies are by default referenced through $A_0$ first; this will make sure that architectural information is maintained aligned at every step of the cycle: modeling $\rightarrow$ transformation $\rightarrow$ modeling and so on.

We will now proceed by presenting in further detail the $A_0$ profile (shown in Figure 3), with a brief introduction to each element. We must stress that, even though we will now focus on our core set of architectural elements, we will keep all the UML 2.0 specification since $A_0$ is a UML profile. The $A_0$ core set is structured as follows:

**SACComponent**: represented as a stereotype to UML components; its main purpose is to identify components on an architectural system description.

**SACConnector**: represented as a constrained stereotype to UML components; its main purpose is that of representing and specifying architectural connectors from an architectural point of view.

**SARelationship**: represented as a constrained stereotype to common UML dependencies; its purpose is that of delineating general relations existing between $A_0$ architectural elements.
**SAType**: type specification can be as important as any kind of topological or architectural specification; SATypes are represented as constrained, stereotyped UML components. Their obvious purpose is that of specifying any aspect concerning a data type definition, which is by itself a particular type of component.

**SATypeSpec**: type specification diagrams will contain the specification of types to correctly and orderly put them in relation or simply to keep them on an architecturally separate level. SARelationships may very well be used among types to define loose interconnection amongst these.

**SACchannel**: channels represent a more general communication means. Because inconsistencies in the design too often procure design flaws, the SACchannel is aptly left general enough so that further specification is categorically needed at a later design time. We originally meant for this artifact to be placed as an association between two generic ports; later in the design of A0 we noticed that the UML super-structure, while it promotes ports from the original property-like usage, it fails in providing ports with their intrinsic semantics of generic interaction-point. We therefore opted for a bi-directional assembly-connector to represent our SACchannel.

**SABehavior**: in most cases an ADL offers some way to specify (and of course analyze) software architecture behavior. State-machines and sequence diagrams come in handy, when it comes to describing architectural behavior with UML. It is for this reason that we chose stereotyped package diagrams to enclose the set of behavioral elements of a system; these elements, specified as behavioral state machines and/or interaction sequence diagrams will be therefore included in the architecture SABehavior diagram.

**SAPort**: quoting from the UML superstructure specification, “a port is a property of a classifier that specifies a distinct interaction point between that classifier and its environment or between the (behavior of the) classifier and its internal parts”. Contrarily to this particular specification we assume a port as being an abstraction of the context (and underlying behavior) in which a certain number of interactions can take place. We re-use the concept of property for that of port specification but it is our belief that it must be enhanced to that of a role classifier, with the possibility of even complex behavior to be attached to it.

In the next section we will explain how the A0 profile interacts within our technology to achieve DULLY’s goal. The previously identified elements from Figure 1 will be analyzed and its interaction point within DULLY will be defined.

**B. DULLY Model Transformation Technology**

To enable the possibilities exposed in the previous sections, precise and powerful transformations must be devised at each level, to “fill” the interconnections existing in Figure 1: model transformations as well as meta-model to meta-model links must be somehow provided. The star architecture permits an agile and easy integration of more and more technologies as the need rises. As previously stated, the transformation system we deliver, is made of a series of low level model to model transformations that enable information migration among model instances. These model to model transformations are constructed automatically by transformations that build transformations, i.e. higher order transformations. Such transformations utilize information provided by the meta-model to meta-model links that “physically” integrate the DULLYzed technology, in the process of creating the lower level
model to model transformations. It is clear that meta-models may also communicate via direct meta-model to meta-model transformations. It may be noted that only the properly skilled meta-modelers will have to take part in the specification section, while software architects can maintain control at their allocated modeling level.

1) **Meta-model to meta-model Links: Bringing meta-models together:** What we earlier called meta-model to meta-model links are the main branches that stem out of $A_0$ and they constitute a mechanism to provide references and semantical relations among elements at a meta-model level and $A_0$; using a terminology more specific to our technology, these semantical links can be called weaving models. As previously stated, weaving models define these links so as to later use them to generate lower level transformations. These links are defined but once during the process that integrates a certain technology within **DUALLy**; their role is in essence that of constituting the bridge between $A_0$ and the **DUALLyzed** technology.

A number of methods to specify and construct weaving models are currently being explored and exploited, but conceptually, they are defined as an instance of a previously existing weaving meta-model, and can be obtained manually, by utilizing scripting languages specific to the purpose. **DUALLy** defines its own weaving meta-model and therefore enables the definition of its own weaving models, which will constitute the main step to be carried out in order to integrate a certain technology.

2) **Higher order transformations: Creating Lower Level Communication:** Lower level model transformations can be constructed just as a model is. Recent research efforts enhanced the concept of model transformation, promoting it into that of transformation model [12]. Just as a model can be created, modified, augmented through a transformation, a transformation can be regarded as a model and therefore, it can itself be instanced, modified and so on. Aply constructed transformations can carry out this particular task: thanks to higher order transformations specified and developed upon **DUALLyzed**, transformations that can migrate information from instances conforming to one meta-model into another, are automatically obtained. Instanced transformations can be executed directly. Unit
all these transformations a network is obtained; this will enable full, level-wide communication.

III. IMPLEMENTING DUALLY

We developed the current version of DUALLY in the context of the ATLAS Model Management Architecture (AMMA) [14]. More specifically, our tool is available as a plugin of the Eclipse platform that extends the ATLAS model weaver (AMW) [20]. We selected AMMA since this model management architecture supports the concepts of weaving model, meta-model independence and it is extensible. Furthermore, models and meta-models are integrated into the same platform and, since AMMA is built on top of Eclipse, they are automatically integrated with several modeling technologies, such as Ecore and UML2. A high-level overview of the technologies we used is represented in Figure 4. Weaving models are particular kinds of models that may be used to define semantic links among meta-models or UML profiles. Both meta-models and models (also weaving models via AMW’s specific editor) are expressed via XMI, while the transformation engine is based on ATL transformations [32]. Implementation details will be described in the following sections.

A. Technologies overview

AMMA is a model management platform designed and developed by the ATLAS Team at INRIA Institute. Its core elements are:

- Atlas Transformation Language (ATL) [32] which is a QVT-like model transformation language, with its own abstract syntax and environment (i.e. an execution virtual machine and IDE). It allows us to transform a source model into a target model. ATL is a declarative and imperative hybrid language. This is one of the key features of ATL: it takes advantage of the simplicity of declarative constructs, while complex tasks (e.g. manipulating UML profiles) can be performed in an imperative fashion. The transformation is itself a model conforming to a specific meta-model. This permits the creation of higher order transformations, i.e. transformations that produce ATL transformations. This feature allows DUALLY to automatically generate ATL transformations as needed.

- Atlas Model Weaver (AMW) [20] is the platform that manages weaving models. It allows the definition of correspondences among models (or meta-models) and to establish semantic links among model elements. The links are saved in a weaving model, which conforms to an extensible weaving meta-model. The weaving meta-model input format is KM3 (Kernel Meta-Meta Model) [31], a language that provides a textual concrete syntax for the coding of meta-models in a Java fashion. The weaving and woven models are defined by the XMI (XML Meta Data Interchange) specification [5], the OMG format for model and meta-models interchange. Besides AMW is featured with a set of extension points that enable us to add specific semantics to the weaving mechanisms.

- AM3 (ATLAS MegaModel Management) is a global resource manager that operates in a model-engineering environment. This component provides support for modeling in the large i.e. dealing with models, meta-models,

3The home page of the DUALLY project is http://www.di.univaq.it/dually while the source code can be found in http://sourceforge.net/projects/dually, released under the GNU General Public License (GPL).
tools, services and their relations as a whole, while ignoring internal details. Within one platform (local or global), a megamodel [15] records all available resources and acts as an MDE repository. From a practical point of view AM3 manages megamodel elements (for example ATL transformations, tools, UML models, meta-models) and provides user interfaces to manipulate them.

- Other peripheral tools are grouped into the ATP (ATLAS Technical Projectors) component. These tools perform model transformation tasks. A subset of ATP tools consists of injectors and extractors to/from other technical spaces (e.g. UML, ATL, XML, SQL etc...) with little or no loss in information.

All AMMA projects are built on top of the Eclipse Modeling Framework (EMF) [2]. AMMA is a modeling framework that provides a MOF-like core meta-model (i.e. Ecore) to define both models and meta-models, tools for importing models and generating code, runtime model support (i.e. reflection, notification and dynamic definition), persistence layer (XML/XMI resource implementations), validation of models and UI-independent viewing and editing support.

B. Realizing the DULLY framework

DULLY is engineered as an extension of AMW. This extension consists of (i) a customized editor for the management of weaving models, (ii) a weaving meta-model that defines the types of link that the user can establish between meta-model elements, (iii) a set of higher order transformations to automatically generate ATL transformations at the model level.

Figure 4 points out how we use the technologies mentioned above. More precisely, both transformations between models and higher order transformations are expressed through ATL. According to the definition of their extension.

Fig. 4. Adopted Modeling technologies
point, weaving meta-models are defined in the KM3 language. The meta-models (or profiles) are expressed in XMI, demanding their import and export to the underlying Eclipse platform. The meta-model level of Figure 4 shows both the intended passageways to integration: the first being the default bridging through the $A_0$ profile and the second being the direct linking between two generic meta-models ($MM_1$ and $MM_2$) holding similar and semantically related elements not contained into $A_0$ itself. From now on we will provide detail only on the former case, leaving aside the implementation of the direct linking, since what is said hereafter can be there re-applied by substituting $A_0$ with a generic ADL meta-model/profile.

The weaving model contains the links between elements of $MM_{x}$ meta-model/profile and elements of the $A_0$ profile. It holds the logic that will guide the automatic generation of the ATL transformations. Each weaving link is used by the higher order transformations (called HOTs in the context of AMMA) to generate either rules or bindings of the ATL transformation. The types of link are specified in the weaving meta-model.

As previously stated, DUALLY operates both on ADL meta-models and UML profiles. The meta-modeler uses the Ecore formalism to define ADL meta-models. These meta-models and their models can be expressed using either the tree-like editor or the graphical editor of EMF. Moreover, many tools exist that import/export Ecore models into/from the Eclipse platform. UML profiles instead are defined using UML2, the implementation of the UML meta-model for the Eclipse platform [6]. The main advantages that DUALLY gains from using UML2 are: compliance with OMG standards (specifically UML 2.0 and MDA) and interoperability with other UML2-based tools. This allows the user to graphically design UML profiles with any UML2-based tool and directly import them into DUALLY. The same mechanisms guide the import/export of UML2 models. By this means our technology achieves independence from tools used for modeling SAs.

As already said in Section II, two are the main users of DUALLY: software architects and meta-modelers. A typical software architect usage scenario is:

1) modeling the software architecture;
2) applying the DUALLY model transformations to the model in order to obtain the software architecture in the target architectural language;
3) working on the software architecture in the target architectural language.

On the contrary, for each meta-model to weave $MM_{x}$ (for example $MM_1$ or $MM_2$ in Figure 4), a meta-modeler usage scenario of DUALLY is:

1) creating (or importing) the meta-model/profile $MM_{x}$ into Eclipse;
2) graphically developing the weaving model between $MM_{x}$ and $A_0$ through the DUALLY weaving models editor (represented in Figure 5);
3) applying the DUALLY higher order transformations to the weaving model in order to automatically generate transformations at the model level.

In this Section we focus and provide details on the meta-modeler’s scenario while the software architect scenario will be explained thoroughly in the case-study section. The first activity of the meta-modeler’s usage scenario...
is based on the import/export mechanisms of the EMF framework; in the following subsections we explain how our technology supports the remaining activities. Section III-B1 presents how DUALLY extends the graphical editor of AMW (mainly by adding the buttons to launch the HOTs and to overcome some presentation issues) and Section III-B2 describes the DUALLY’s weaving meta-model. The last two subsections explain the higher order transformations of DUALLY (Section III-B3) and the generated first-order transformations (Section III-B4), respectively.

1) DUALLY weaving models editor: The weaving models editor is DUALLY’s graphical front-end: it manages the interaction between users and the model transformation engine. It is composed of three main panels: weaving panel, left woven meta-model panel and right woven meta-model panel.

In Figure 5 we illustrate how the weaving model and woven meta-models are represented within the DUALLY’s editor. The weaving panel (part B in Figure 5) is a tree-based editor to create a weaving model, i.e. the mappings between elements of two meta-models (left and right). It is built on the base weaving panel provided by AMW, taking advantage of the reflective EMF capabilities; this avoided us the effort of writing a specific editor for DUALLY and enabled us to rely on a tested editor also commonly used in research.

Fig. 5. Graphical interface of DUALLY

The elements of the editor reflect those of the weaving meta-model (later described in Section III-B2); each element is featured with its own icon and is created through a contextual menu. These choices speed up and simplify the development of weaving models: if the user clicks on a given element, the contextual menu shows
only the child and sibling elements that may be added to such element, in respect to the rules imposed by the meta-model.

The basic toolbar of the weaving panel of AMW was extended with two buttons (see part E in Figure 5) that allow the user to execute the HOT directly from the weaving panel; DUALLY automatically retrieves the information needed to launch the transformations.

The left and right woven meta-model panels (parts A and C in Figure 5) graphically represent left and right meta-model, respectively. These panels extend the reflective editor provided by EMF with the aim of representing the meta-models/profiles as hierarchical trees. The extension solves a problem presenting itself when dealing with hierarchical meta-models: suppose that the user is creating a weaving model between UML profiles, namely leftProfile and rightProfile; a correspondence between the Port element of leftProfile and the Interface element of rightProfile must be created. At this point the user has to specify also the binding between structural features, say the “name” attributes of both sides. If DUALLY uses the standard EMF tree editor, the Port and Interface elements contain only their own features (leaving out the inherited ones, e.g. “name”), so the user has to navigate the tree searching the feature “name” in the parent elements of Port and Interface. This makes the creation of weaving links complex and error-prone. We extended the EMF tree editor so that each element displays all its structural features (inherited and local).

The decision to render the meta-models as trees was taken according to three main reasons: (i) the meta-models that formalize ADLs usually contain a large number of elements (e.g. UML profiles) and a graphical viewer could make DUALLY too complex to use, (ii) meta-models usually contain many hierarchical relations and the tree-view best shows this feature, (iii) to render models as trees is a common practice in the modeling field, users will require less training to use DUALLY.

Woven meta-model panels have a toolbar (part D in Figure 5) that allows the user to render the meta-model (profile) as an Ecore diagram (UML profile diagram) taking advantage of EMF and UML2 Eclipse plugins.

As mentioned earlier, the meta-models to weave form an $A_0$-centered star topology, therefore either left or right woven meta-model panel should contain $A_0$.

Particular attention has been paid to the management of UML profiles. The procedure that DUALLY automatically applies while loading a UML profile is composed of two main steps:

1) transform the profile into an Ecore meta-model. In so doing, we avoid to develop specific HOTs from Ecore meta-models to UML profiles and viceversa. More precisely, we develop only two HOTs, one handling transformations from Ecore to Ecore and the other handling transformations from UML profiles to UML profiles.

2) add the UML meta-model package to the profile. This step is unavoidable because the XMI file of a UML2 profile contains only the extensions defined by the profile. Conversely, a profile is by definition an extension of the UML meta-model, and so it must contain both its elements and the elements of the UML meta-model. DUALLY programmatically adds the UML meta-model to the profile to fill this gap. We remark that the UML meta-model is not hard-coded in DUALLY, but rather it is dynamically retrieved from the UML2
Eclipse plugin [6]. Therefore, **DUALLY** is auto-upgrading with respect to future UML versions.

In so doing, **DUALLY** becomes an integrated environment to map ADL constructs, to graphically develop the corresponding meta-models (profiles), and to perform model transformations of all possibly instantiable models. Furthermore, this environment is open to other modeling tools thanks to the importing mechanisms provided by the EMF Eclipse platform.

2) **DUALLY weaving meta-model**: The AMW Eclipse plugin is based on a small weaving meta-model, defining the abstract notions of weaving links (what we earlier referred to as meta-model to meta-model links) between model elements.

This meta-model is abstract and is thought to be as generic as possible. However it provides an extension mechanism to define domain-specific types of link. We developed **DUALLY**’s own weaving meta-model by extending the AMW basic one. Our extended meta-model adds ADL-specific constructs (e.g. specific links to map structural features) and is oriented to the automatic generation of ATL transformations (e.g. there is the distinction between source and target woven elements).

![DUALLY weaving meta-model](image)

**Fig. 6. DUALLY weaving meta-model**

Figure 6 illustrates the **DUALLY** weaving meta-model through its Ecore diagram (a graphical formalism to define Ecore-based models [2]).
The **DUALLY** weaving meta-model elements are:

- **DUALLY**: it extends WModel of AMW and is the root element of each weaving model. It is composed of the references to **left** and **right** meta-models and a set of correspondences.
- **Correspondence**: it represents a generic mapping between elements of the woven meta-models. It extends the WLink element of AMW augmenting it with the `condition` property. `Condition` specifies the ATL guard of the matched rule to generate and is automatically injected into the generated ATL transformation. A `Correspondence` can be of three types: **Left2RightCorrespondence**, **Right2LeftCorrespondence** and **EquivalenceCorrespondence**. This is a peculiarity of **DUALLY**: it adds navigability semantics to weaving links. This gives us many advantages: (i) easier automatic generation of different ATL transformations from the same weaving model, (ii) creation of more complex and structured weaving models, (iii) strict relation between each weaving mapping and the corresponding generated ATL rule.

  - **Left2RightCorrespondence**: this element represents a unidirectional correspondence, specifically a correspondence from an element of the **left** meta-model to one or many elements of the **right** meta-model. It contains a reference to one and only one element of the **left** meta-model and a reference to one or many elements of the **right** meta-model.
  - **Right2LeftCorrespondence**: this is the specular version of the previous element. So it contains a reference to one and only one element of the **right** meta-model and a reference to one or many elements of the **left** meta-model.
  - **EquivalenceCorrespondence**: this element represents a correspondence with bidirectional navigability. The woven elements of this correspondence are exactly two (one from the **left** meta-model and one from the **right** meta-model), they are equivalent from the weaving point of view. This element contains a set of `FeatureEquivalences` that represents the bindings between the structural features (i.e. attributes and references) of the woven elements. It contains also a boolean attribute called `MatchAllFeatures` that will be explained in the next section.

- **WovenElement**: it extends AMW WLinkEnd and it is the abstract element that indicates the extremity of a correspondence. The feature `variableName` represents the name of the variable assigned to the element in the generated ATL transformation. Similarly to `Correspondence` there are three types of **WovenElement**: **SourceElement**, **TargetElement**, and **EquivalenceElement**.

  - **SourceElement**: it is the source element of a directed correspondence. For example a **Left2Right-Correspondence** must contain a **SourceElement** belonging to the **left** meta-model.
  - **TargetElement**: it is the target element of a directed correspondence. Similarly to **EquivalenceCorrespondence** it contains a set of `FeatureEquivalences` and the `MatchAllFeatures` attribute.
  - **EquivalenceElement**: it represents an element of an **EquivalenceCorrespondence**.

- **FeatureEquivalence**: it extends AMW WLink and defines a mapping between two features.

- **Feature**: it extends AMW WLinkEnd and represents a structural feature of a woven element.

- **WovenElementLink**: it extends AMW WLinkEnd and specifies a correspondence between a **WovenElement** (or one of its features) and a feature of another woven element.

3) **DUALLY** higher-order transformations: **DUALLY** HOTs act as a bridge between the meta-modeling and modeling levels. Figure 7 illustrates the configuration of **DUALLY** while generating ATL transformations between
MM1 and $A_0$. The other symmetric transformations are generated with the opposite configuration.

The input of a HOT is composed of three models: (i) the weaving model, (ii) the left meta-model and (iii) the right meta-model. The output is an ATL transformation generated on the basis of the mappings defined into the weaving model; at a high level of abstraction, the current version of DUALLY translates each Correspondence into an ATL matched rule and each FeatureEquivalence into an ATL binding.

Furthermore, bindings can be also generated implicitly by the MatchAllFeatures attribute in a TargetElement. If this attribute evaluates to true the higher-order transformation calculates the union of the structural features of the woven elements, determines the subset of features with the same name and automatically generates the ATL bindings between the features of the subset. This avoids the manual definition of numerous weaving links when all the features of two elements must be bound.

If the user sets MatchAllFeatures to true and defines other feature equivalences, these will override the corresponding implicit ones, making the editor very flexible. For example, suppose the existence of $elA$ and $elB$, in the left and right meta-models, respectively. $elA$ and $elB$ contain ten attributes with the same name. Figure 8 illustrates two versions of a weaving model that binds all the features of the elements. The weaving model (a) contains a feature equivalence for each feature to map and (b) implicitly binds such features. The weaving model (a) has a complete control on the bindings, but the weaving model (b) is more readable and easy to design.

There are two HOTs in DUALLY: Left2Right and Right2Left. The former generates a transformation from models conforming to the left meta-model to models conforming to the right meta-model while the latter generates the inverse transformation.

DUALLY’s HOTs also perform semantic checks. For example we faced a typical MDE problem during initial
testing phases of the tool, the “round-trip problem”. Considering Figure 4, let us assume that the ATL transformations are already generated. Let us also assume that M1 makes a “round-trip” journey into \textbf{DUALLY}; in other words M1 is transformed in M(A_0), then in M2, again to M(A_0) and finally in the initial modeling technology. There are two possibilities now: (i) the newly obtained M1 is equal to the initial one or (ii) it is different. The HOTs of \textbf{DUALLY} analyze the weaving model verifying if it is designed so that the second case would happen and show a warning message to the user (specifying the correspondences affected by this problem).
The “round-trip” problem never affects weaving models composed only by equivalence correspondences; indeed they specify equality relations between elements and do not set up a directed “flow” of modeling entities.

4) **DUALLY generated transformations**: DUALLY automatically generates ATL transformations at the model level; we call them Basic Transformations (BT). For example these transformations are represented in Figure 4 by the arrow between M1 and M(A0).

The main difference between HOTs and BTs is that the former ones operate at the meta-model level while the latter ones operate at the model level. This difference reflects on the quantity of executions of each kind of transformation: a HOT is usually executed once for each meta-model MMx during the generation phase; on the contrary BTs are executed on each model conforming to MMx.

The HOTs dynamically verify if one of the input meta-models is a UML profile and reflects it to the generated BTs. In fact there are four types of generated transformations, depending on the type of meta-models they are generated from (Table I). The element Ecore represents a MOF-compliant ADL meta-model, UML profile represents a UML profile, UML represents the UML meta-model and PRO represents the MOF definition of a UML profile. The last element is necessary because the ATL transformation must retrieve the definition of the stereotypes and tagged values that may have been set in the target model.

<table>
<thead>
<tr>
<th>Source meta-model</th>
<th>Target met.</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ecore</td>
<td>Ecore</td>
<td>Ecore</td>
<td>Ecore</td>
</tr>
<tr>
<td>Ecore</td>
<td>UML profile</td>
<td>Ecore</td>
<td>PRO</td>
</tr>
<tr>
<td>UML profile</td>
<td>Ecore</td>
<td>UML</td>
<td>Ecore</td>
</tr>
<tr>
<td>UML profile</td>
<td>UML profile</td>
<td>UML, PRO</td>
<td>UML</td>
</tr>
</tbody>
</table>

**TABLE I**
**INPUT/OUTPUT OF THE GENERATED TRANSFORMATIONS**

Since the specification of an ADL through a UML profile is widely used in industry (e.g. [28], [36]), much attention is paid to the process of weaving two UML profiles (see the last row of Table I).

ATL allows us to bind any number of transformations on top of each other, establishing hierarchical relations between them. More precisely, the generated BT contains only rules that transform stereotyped elements. At this point the BT will be superimposed on a fixed transformation (embedded into DUALLY) that contains the copying rules between standard UML elements. The task of this transformation is to copy all UML2 model elements from the input model to the output model. The generated transformations inherit all the rules (they are always the same) that transform UML standard elements. This inheritance implies that: (i) the generation process is more agile, since DUALLY does not generate the approximately two hundred rules otherwise needed; (ii) the BTs are more readable since they do not contain verbose copying rules; (iii) the debug of BTs is straightforward because there is a 1:1 relation between each weaving correspondence and the generated rule.

The copying transformation is itself automatically derived from another (tested and used in research) ATL transformation [4].
We faced other technical issues while generating transformations that handle the UML2 Eclipse meta-model. This is caused by the specification of UML models provided by the Eclipse UML2 platform. UML2 is a MOF-compliant model, but the profile (stereotype) applications are only annotated to the elements. This means that the applied stereotypes of an element are not directly visible, but must be accessed through UML2 specific method calls.

Figure 9 shows an example of a generated ATL rule that transforms a UML component into a stereotyped one.

```plaintext
rule DComponent SComponent {
  from s : UML2!Component 
  {thisModule.inElements->includes(s) 
    and s.isStereotypeApplied('DComponent')} 
  to t : UML2!Component mapsTo s ( 
    name <- s.name, 
    visibility <- s.visibility, 
    ownedReception <- s.ownedReception, 
    packageElement <- s.packageElement, 
    realization <- s.realization) 
  do { 
    t.applyStereotype(thisModule.SComponentStereotype); 
  }
}
```

Fig. 9. Example of a generated ATL rule.

The source element of a rule is matched against the metaclass of a stereotyped element, while the condition of the rule verifies if a specific stereotype is applied. The same mechanism is applied to the target element of the rule. Similarly an ATL rule cannot bind the tagged values of a stereotype, but must get/set their values through UML2 specific method calls.

DUALLY includes a sub-system that manages the name of variables in the BTs. It prevents the creation of transformations with conflicting variable names; if the user does not specify the attribute `variableName` for a woven element, DUALLY also generates a unique identifier for such element.

IV. IECS Case Study

In this section we show how the conceptual features of DUALLY are applied to a real case study, explaining the typical usage session of our framework from the point of view of a software architect. Figure 10 depicts the modeling technologies we used to develop the case study.

Among the notorious ADLs we selected the DARWIN/LTSA modeling and analysis environment and the ACME ADL. We choose these ADLs for the presentation’s sake since these ADLs are not so different and thus it is easy to show their DUALLYzation as well as the translation of a model designed in Darwin into an ACME specification. In Section VI we will show our experiment of DUALLYzation of AADL that is, on the contrary, somewhat different than either of these. Upon this experiment we will argue about how each ADL can be DUALLYzed and how even most different ADLs can be successfully related with each other.
There are essentially two ways to create the extension by means of which $A_0$ and any ADL can communicate with each other:

1) recreate the meta-model representing the ADL’s syntax and semantics, which can be done using EMF, instances of EMOF or MOF itself;

2) designing the profile reproducing all of the ADL’s elements and structures, deriving directly from the UML 2.0 standard as its reference meta-model.

In this use case, we have chosen to dwell in each of these two approaches trying in the last part of this section to explore the underlying implications thereto. Therefore Darwin/LTSA is expressed by a UML profile and ACME is represented by an EMF-based meta-model.

The whole process of the case study can be divided into two main phases, operating at the meta-model and model level respectively:

1) the MDD expert develops the weaving models (points 1 and 2 in Figure 10) and DUALLY automatically produces the following ATL transformations: Darwin2A0 (point 3 in Figure 10) that produces $A_0$-stereotyped UML models from models conforming to the Darwin/LTSA profile, and A02Acme (point 4 in Figure 10) that takes as input the model generated by Darwin2A0 and produces the corresponding ACME specification;

2) software architects execute the generated ATL transformations obtaining ACME specifications (7) from Darwin models (5) and vice versa; in this case $A_0$ models (6) are the means by which the two architectural languages interoperate.

Next sections present the details of the case study.
A. DARWIN/LTSA and ACME meta-models

We modelled Darwin standard syntax and semantics in a regular UML 2.1 profile, represented in Figure 11. References for the development of DARWIN/LTSA profile were taken from [29]. In particular this profile contains both the structural elements of Darwin and the behavioral constructs of LTSA. It is natural to map a Darwin specification into a stereotyped UML component diagram, while each behavioral LTS (Labeled Transition System) is expressed via stereotyped UML state machines. Specific semantics of the two languages (i.e. Darwin and LTSA) were implemented by stereotyping UML elements and adding tagged values to them. For example an LTSTransition has an ActionType attribute indicating if it is either a tau (i.e. internal) or a shared transition.

For further details of the elements represented in the profile please refer to [29]. The profile needs only to be exported to EMF standard code and is ready to serve within DUALLY.

We did not develop a DUALLY-specific ACME meta-model, but we chose to adopt one which is already used in research as part of the AMMA initiative. This ACME EMF meta-model can be found within AMMA project zoos in [1] as part of a demonstrative transformation rendered available by the ATLAS AMMA research group. Our case study is based on the simplified ACME meta-model represented in Figure 12.

An ACME specification is represented by an ACMEFile element. This element is composed of a set of ACMEEntries. There are two possible entry types: System and Component Type. The first contains the specification of the architecture, while the latter represents a type of architectural element. Other entities represent the seven basic types establishing the core ontology of ACME: component, connector, attachment, representation, property, port and role. Further conceptual details on the ACME meta-model can be found in [9].
B. Weaving Models: Darwin_A0 and A0_Acme

The weaving model is the mean by which the software architect establishes semantic bindings between two ADLs. Therefore, in our case study we developed two weaving models:

1) Darwin_A0 specifies the bindings between the Darwin/LTSA and A0 profiles;
2) A0_Acme contains the bindings between the A0 profile and the ACME meta-model.

The weaving models presented in this section contain only the minimum number of correspondences to generate the needed HOT, leaving the models as readable and understandable as possible.

Figure 13 represents a simplified version of the Darwin_A0 weaving model. For the sake of simplicity we focus only on the structural elements, abstracting from the behavioral aspects of the woven profiles.

The semantics emerging from this weaving model are:

- each DAcomponent of Darwin/LTSA profile corresponds to an SAcomponent of A0;
- a DAbinder corresponds to an S Achannel;
- each provided DAinterface corresponds to two standard UML elements: an Interface and an InterfaceRealization that connect them to the Component the DAinterface belongs to;
- each required DAinterface corresponds to an Interface and a Usage relation in the same manner of the previous point.
All the correspondences of the weaving model have the \textit{MatchAllFeatures} attribute set to \textit{true}; so the framework infers the common features of each couple of source/target elements and maps every structural feature of the source element to the target feature with the same name.

Figure 14 represents a simplified version of the $A_0$-	extit{Acme} weaving model.

The relevant correspondences of $A_0$-	extit{Acme}:
- a UML model is translated into an ACMEFile that contains an ACME System;
• each SAcomponent corresponds to an ACME ComponentInstance;
• an SAconnector corresponds to an ACME connector;
• each UML property corresponds into an ACME Property;
• an SAchannel corresponds to an ACME Connector containing two predefined Roles. Each role corresponds to an auxiliary ACME Attachment. These additional elements are necessary because ACME component instances interact only through a Connector;
• a Port contained into an SAcomponent corresponds to an ACME Port;
• a Port contained into an SAconnector corresponds to an ACME Role.

C. Generation of ATL Transformations

The weaving links form the logic to generate the ATL transformations of our use case. They suite well to the models of our case study, constituting at the same time a solid starting point for their refinement towards more structured ATL transformations.

The execution of the HOTs constitutes the transition from the meta-model to the model level; indeed they take in input models of meta-model level and return transformations operating at the model level. The generation phase is the most crucial (for the issues described above), but our framework makes it totally transparent to the software architect who has only to press the corresponding button of the tool.

```plaintext
61.§-- GENERATED
62.rule DBinder_SAchannel {
63   from
64   s_4 : UML2Dependency
65   {
66     s_4.isStereotypeApplied('DBinder')
67     and thisModule.inElements->includes(s_4)
68   }
69   to
70   t_8 : UML2Dependency {
71     owningTemplateParameter <- s_4.owningTemplateParameter,
72     eAnnotations <- s_4.eAnnotations,
73     client <- s_4.client,
74     visibility <- s_4.visibility,
75     templateParameter <- s_4.templateParameter,
76     nameExpression <- s_4.nameExpression,
77     name <- s_4.name,
78     ownedComment <- s_4.ownedComment,
79     clientDependency <- s_4.clientDependency,
80     supplier <- s_4.supplier
81   }
82   do {
83     t_8.applyStereotype(thisModule.SAChannelStereotype);
84   }
85}
```

Fig. 15. Excerpt of a generated transformation.

Our case study contains two generated ATL transformations:

• **Darwin2A0**: it is obtained from **Darwin-A0** weaving model and returns **A0**-profiled models from Darwin/LTSA models;
ACME2A_0: it is generated from A_0_Acme and returns ACME specifications from A_0 models.

The semantics of each transformation reflects the correspondences contained into the weaving model it is generated from. For example Figure 15 shows an excerpt of Darwin2A_0.

In particular Figure 15 presents the rule that transforms a DAbinder into an SACHannel along with the bindings of their common features automatically created by DUALLY. Notice that, conforming to the UML2 Eclipse specification, the target element is a standard UML Dependency and only later the SACHannel stereotype is applied through the method call ApplyStereotype.

D. IECS Modeling in Darwin/LTSA and Transformations to ACME

The software architecture of our case study models a multi-tier environment capable of maintaining a fail-safe, client-server like communication within a safe and secure environment such as a military vessel: the Integrated Environment for Communication on Ship (IECS) [16]. The case study’s specification comes from a project developed within Selex Communications, a company mainly operating in the naval communication domain.

Fig. 16. Static description of IECS-MS architecture using Darwin/LTSA profile.
Because of the case study size, we chose to consider a sub-set of its architecture so as to keep both the application steps and the further results consideration part as user-friendly as possible. The subset was obtained by isolating the main part of the actual system itself: IECS-MS, IECS Management System.

Figure 16 shows the overall structure of IECS-MS modeled by using the Darwin/LTSA profile.

The IECS-MS has a Manager-Agent architecture that allows us to administrate the load balancing. The hardware configuration is composed of four main servers (identified by the term Master), they are:

- **master_main_console** that synchronizes and activates the communication services;
- **master_CTS_proxy** that handles the connectivity provided by another component called CTS;
- **master_UMF** that manages other multi functional component units, each called UMF⁴;
- **master_DB** that stores data and configurations.

Every master component is replicated by an equivalent Slave server in order to guarantee a certain level of

⁴The acronym comes from the Italian “Unità Multi Funzionale”.

March 31, 2008 DRAFT
reliability. equipment_1 and equipment_2 represent generic components of the architecture. The system contains also one additional_console used by the administrator to access the system. proxy_computers are simple workstations that provide auxiliary communication facilities to the components of the system. The connectivity between servers, consoles, proxies and other equipments is provided by an Ethernet LAN, while the components that have a proprietary interface (e.g. UMF, CTS) communicate via serial lines.

Once the model illustrated in Figure 16 has been created, we give it as input to Darwin2A0 (the transformation automatically generated by DUALLY). Figure 17 shows the model resulting from such execution.

The first consideration that needs to be made regards the source and target model graphical representations. They are different because we want to emphasize how our framework achieves the modeling tool independence. The idea is to create the source model using a specific tool, apply the transformations and later import the generated model into another UML-based tool. The selection of the modeling tools is based exclusively on the power of their EMF import/export modules. Therefore we developed the source model (i.e. Darwin/LTSA profiled IECS-MS architecture) using NoMagic’s MagicDraw UML modeler, while the target model (i.e. A0-profiled IECS-MS) has been imported into Visual Paradigm for UML.

![Source state machine in Darwin/LTSA profile](image1)

(a) Source state machine in Darwin/LTSA profile

![Target state machine in A0 profile](image2)

(b) Target state machine in A0 profile

Fig. 18. Transformation of behavioral specifications in DUALLY

Other kind of considerations must be made on the semantic differences between the source and target models. The topology of the target model is equivalent to the topology of the source architectural specification. The generated transformation promotes a UML property (i.e. each port of a component) into a set of structured elements: an Interface and an InterfaceRealization (or Usage). Therefore, we demonstrate that our framework allows us to promote or demote architectural elements, according to the bindings defined into the weaving model.

March 31, 2008 DRAFT
Regarding the Ethernet component of the target model, a software architect might expect that it is represented via an SAconnector, but the lack of the concept of architectural connector in Darwin forces the BT to transform it into an SAcomponent. Indeed conforming to a well-known principle of MDE, our BT cannot add semantics to the target model.

The case study includes also a behavioral description of the IECS-MS architecture, but for the sake of simplicity we leave the details out of this paper.

Figure 18 gives an idea on how the behavioral models are transformed: section (a) represents a sub-machine of the behavior of IECS-MS in Darwin/LTSA, section (b) illustrates the corresponding state machine generated by the ATL transformation.

The final phase of the case study corresponds to the transformation of the current $A_0$-profiled model of IECS-MS into an ACME specification. The result of this transformation is the EMF-based model represented in Figure 19.

Analyzing the ACME specification, we can affirm that it is isomorphic to the model in $A_0$. For the sake of simplicity we did not specify bindings targeting the concept of ACME family. Therefore, the generated ACME specification contains only base (i.e. non-typed) components.

Because EMF ecore meta-modeling presents no box-and-lines editing notation and no graphical front-end up to the present day, the software architect will discover that EMF notation is in its part, less readable, and prone to misinterpretation as well as lack of formality. To overcome this problem an ATL transformation that takes as input the Ecore model of an ACME specification and returns the textual specification conforming to standard ACME syntax may be developed.
V. RELATED WORK

In the following section we will proceed and delineate the ADLs as well as other technologies, frameworks and other research attempts in DUALLy’s direction, which we took as an inspiration point while working on DUALLy itself. Our main intent is that of showing a glimpse of existing technologies, their particular and personal domain as well as their own support for analysis and other design feats. From these considerations we derive the rationale behind DUALLy.

The length of the section will be organized in two parts. In Section V-A we will provide insight on a number of ADLs whose purpose we considered closer to that of DUALLy: the ACME initiative [26] from Carnegie Mellon; the xADL project [18], a very effective and extendable description framework and AADL, an industrially adopted and affirmed architectural description framework originally meant mainly as an avionics-focused DSL. In Section V-B we will mention other attempts as well as other closely related efforts at tackling the problems encountered and surpassed by DUALLy.

A. Sightseeing related ADLs.

The ACME initiative [26] is famed for being one of the very first technologies to tackle the problem of architectural data interchange. ACME and its supporting tool ACMEStudio were born as a simple, multi-style ADL framework also providing the possibility of using it as a common interchange platform for multiple ADLs. It eventually drifted away from this particular field now standing on its own as an architectural description and analysis framework.

ACME provided foundations and mechanisms to extend the framework. Tooling extension points come from allowing other tools to physically read and write ACME descriptions while semantical extension is enforced by allowing properties to carry ADL-specific data within the model. The provided ACMElib library can also be used to adapt ADL technologies to ACME and to allow their manipulation within ACMEStudio. Some additional efforts brought ACME closer to UML by providing a ready-made profile for this task [28]. A number of reasons can be presented for ACME drifting somewhat away from its initial goal, the main of which is that its core technology does not provide direct support to integration with other architectural description frameworks. Moreover, efforts invested in integrating technologies within ACME are not reusable outside of ACMEStudio [26]. Descriptions instantiated within ACME will still have problems in “exiting” the technology and will depend heavily on its intrinsic format. Moreover, industries using already existing ADLs, will inevitably have to shift modeling paradigm: none of the modeling knowledge and insight of the previous modeling frameworks will be easily reusable; integrating ACME presents no industrial roll-out from the “legacy” modeling technologies.

DUALLy keeps the idea of a base kernel of elements acting as doorway for information migration. Contrarily to ACME, DUALLy offers the possibility of modeling directly with simple UML or UML augmented with other specific profiles. By means of fully automated transformations, DUALLy will allow for semantic information to be maintained and upheld. From an interchange specific perspective, DUALLy renders it simple to integrate a technology while keeping it core-independent, thanks to the mechanisms of profiling and meta-modeling used upon integration.
Lastly, by relying heavily on UML knowledge and skills (i.e. modelling, meta-modelling and profiling) **DUALLY** becomes sensible to the industry’s perspective of skill reusability: development teams will no longer have to re-qualify. Using **DUALLY** will be as simple as re-using previously acquired knowledge and skills on profiling and meta-modelling.

**xDADL** [18] is an active research effort born and in progress at ISR - Irvine University. Just like **DUALLY**, the technology bases itself around a core of elements as a reference: xArch. xDADL inherits from xArch a number of like-to-have features such as direct run-time instantiation of SAs, model grouping, SA hierarchy and so on. xDADL as well as its core xArch, are based on XML and thus fully extendable [17]. The xDADL project evolved into a powerful technology backed by a solid workbench, ArchStudio 4, which maximizes the possibilities the technology shows. Considerable efforts were poured in developing XTEAM [34]: an MDE (Model Driven Engineering) focused extension to the xDADL framework. Such an attempt shows the importance to support and integrate MDA (Model Driven Architecture) [3] and MDE-focused industrial development processes. The possibilities and limitations exposed by xDADL on its own are very similar to those evidenced within ACME. Integration efforts have still to physically augment the common XML architectural description provided within xArch: in this sense, a typical scenario would imply extending the core once for one technology and yet again for another one. The same core will become bigger with the first extension and yet even bigger because of the second one: this will still make the extensions “depending” upon one and the other.

Following these points, a natural comparison is possible between xDADL and **DUALLY**: our approach is very close to the strong-points and novelties introduced by XTEAM as evidenced in [34], and it also provides full compatibility and support to the MDA process, as recently standardized by OMG in [3]. While xDADL may be used to define DSLs, i.e. it is a structured language engineered to be extended and “configured” so as to represent desired domain specific architectures, **DUALLY** may be seen as a multi-technology interchange language, i.e. a framework and an infrastructure to allow intercommunication and architectural data migration between ADLs themselves and/or UML profiles. Where xDADL promotes extensions at a low level, **DUALLY** utilizes profiles and meta-models for the implementation of lightweight and heavyweight integrations. **DUALLY** and xDADL, however, both share a modular structure and hence neither can be seen as a single monolithic block within which to construct software architectures.

**AADL** [22] was born as an avionics focused DSL and later on moved to representing and supporting embedded real-time systems: its name and acronym were then re-factored to architecture and analysis description language. Standardized by the SAE (Society of Automotive Engineers) AADL now shows impressive usage in the field of both software and hardware specification and validation. AADL was designed as an extensible core language supporting modeling from multiple aspects and points of view. The validation technology it provides, addresses timing and performance properties of systems. Concurrency and interaction semantic specification is enforced. AADL was also one of the very first ADLs that introduced the concept of “feature” as a component’s way to interact with others at the implementation level. Moreover, the key concept of “aspect” of a system, during its
development, is also available within the technology as exposed in [19]. The notation’s extension mechanisms include the definition of custom properties to specify additional ADL-specific analyses and/or generic information to be attached on the architectural design. Additional notation extension efforts are bringing AADL closer to UML friendly notations (via a profile) and are in their final steps [23]: the initiative has developed a UML profile (or rather an xUML profile, i.e. UML with formal action semantics embedded in it) to later on synchronize the two technologies. In addition, the main open source tool of the technology, OSATE tool, is very close to our view of an ideally extensible framework: it supports plug-ins, core-set extensions and might be made to support MDA specific technologies, by coordinating it with the Eclipse MDA initiative. Other extensibility mechanisms defined through constructs such as the standard “annex” plus the mentioned property set extensions are closely related to our view of “semantic” enhancement, which we consider very important in any technology. Our research effort took AADL as a key reference due to the exposed features and also because of its widespread adoption within dependability critical industrial development processes. Even in this case, it must be noted that the technology fails in providing automated support to its extensibility possibilities. Somewhat like xADL, AADL can be viewed as a modeling notation that can be complemented by description technologies tailored to specific goals of a particular modeling view and it can be considered a language supporting DSL generation. We decided to follow other recent trends and research directions in the ADL field confirming that interoperability between these domain specific realities is still to be realized effectively. As a consequence, DUALLY’s goal is that of providing automated interoperability and synchronization mechanisms between technologies while also making no compromises concerning user-friendliness.

B. Related Efforts.

DUALLY’s basic architecture is centered around a core set of essential descriptive elements: \( A_0 \). This idea derives primarily from the requirement of providing a common minimal ground language to enable all architectural description frameworks to “communicate” between one another. This communication is a mid-way layer between every integrated technology. Previous technologies and some UML-based attempts at integration or architectural description, underlined the need for a basic set of meta-elements to be used as a reference set. A number of different yet close attempts both at the description and formalization of such a set may be found in [42], [33], [30] and [10].

In [33], Kandé and Stromheier point out the need for a more specific approach of UML to the issue of effective and complete description of software architectures. They then define a UML profile trying to tackle such an issue. Their profile proposal figures up as a view-point oriented specification of software architectures by means of pure UML and additional constrained constructs. This viewpoint-centric approach does not touch some common issues in the field such as SA static validation, architecture and style verification, architectural drift avoidance (i.e. maintaining the architecture synchronized with its lower-level artifacts) etc. All these issues are put aside considering the multiple migration possibilities granted by DUALLY: a design element can be ported into different validation environments or different diagram representations. Viewpoints themselves are grafted within the very core of DUALLY: it is possible to use a number of different architectural concepts in different diagrams and diagram groupings so that the description is kept consistent and separation of concerns is maintained. Moreover,
DUALLY’s high scale extensibility will allow for the integration of enhancements such as viewpoint-validation, style consistency check and so on.

In [42], a UML 2.0 focused architectural description enhancement, conceptually close to DUALLY and A₀ can be found. The paper describes in fact, a set of core elements that must be enhanced within UML itself so that appropriate and accurate architectural description can take place. Many of the inconsistencies there identified are taken out within our approach and almost all of the proposed enhancements are already in place or easily implemented within further developments on DUALLY itself, or more specifically, on the A₀ core set: extensions, tailoring and customizations of the A₀ profile are possible through the simple mechanism of profiling, now becoming increasingly common in many industrial environments. The level of architectural constraining can be decided and augmented at will, by simply re-touching the A₀ core set and the related transformation engine.

In [30] we may find a perfect example of meta-information exchange between ADL formats via model transformation. This particular case shows a glimpse of the full potential that can be achieved within DUALLY, its transformations and our transformation engine. Considered technologies are ACME and META-H⁵, two of the previously mentioned mainstream modeling technologies. The paper shows in detail the actual implementation of the meta-interchange to take place from ACME to META-H, as it is carried out within the eclipse framework. The same principles are applied within DUALLY with the chief difference that the shown transformations will execute mainly on model instances whereas DUALLY is able to encompass a much wider scope. In fact, DUALLY’s interchange engine will make sure that semantic communication is on place on both modeling and meta-modeling level.

In [10] Smeda et al. bring around a relatively new concept in the field of architectural description: that of meta-architecture description language, i.e. Meta-ADL (MADL). The paper focuses on the specification of a meta-level to describe ADLs right from their core level, rather than architectures themselves. The paper shows the undoubted potential behind the specification of this meta-meta-model for architectural description languages, and provides a possible implementation of such a technology.

The approach however shows some serious shortcomings: the development of a totally stand-alone architectural “MOF” is both expensive in terms of effort and compatibility with UML. Utilizing such a technology in fact, would mean to sacrifice compatibility with UML right from its MOF. We assume this incompatibility as being unacceptable in the modern architectural description field we previously presented. DUALLY on the other hand, while being on the same concepts of MADL, provides full compatibility with both eclipse and UML (EMOF). [10] also provides a detailed assessment of architectural description issues that are yet to be solved (such as full standardization, architectural comparison, integration and interchange of formats, styles etc). While MADL effectively tackles these issues, some newly opened ones remain unhindered: industrial integration, technology roll-out, knowledge re-use and a number of other issues will remain untouched. DUALLY tries to work on the same scope as in [10] while also trying to “think industry” by considering the industrial prospects and problems on architectural description.

⁵http://www.htc.honeywell.com/metah/prodinfo.html
DUALLY, in essence, fully maintains the OMG concept of modeling-information (level 3. MOF - level 2. UML - level 1. Models - level 0. instances) by completely installing itself and providing support at each of these levels.

VI. EVALUATION AND CONSIDERATIONS

In this section we provide an evaluation and some considerations about DUALLY. The discussion will be organized in four main arguments: (i) quality of the generated transformations, (ii) extendibility and interoperability among ADLs and SA UML profiles, (iii) masking the complexity and (iv) integration with MDA technologies.

(i) Quality of the generated transformations: a transformation process is composed of three parts: source model, target model and transformation itself. Assuming the correctness of the source models, when discussing about quality of the generated transformations we deal with quality of the obtained target models. In DUALLY, the target model natively conforms to its meta-model (i.e. the quality of the transformation creating the target model does not count as the model will be, by construction, an instance of its reference meta-model). Obviously, if the meta-model is not accurate and completely correct, this may hinder the transformations quality. Dealing with transformation quality requires to ensure that certain “properties” are preserved by transformations; more precisely, if the source model satisfies a given set of architectural properties then the target model has to satisfy those properties as well. Since we are considering properties that are intrinsically intermingled with a specific system we can only provide methodologies and guidelines to ease the problem’s solution.

Based on [13] two are the main approaches to tackle the problem: a Generic approach and a Transformation-based approach.

Generic approach. As represented in Figure 20.a, two are the input models: the model to test and the model containing the specification of the properties to verify. The Verifier component may be implemented as a generic ATL transformation, so we can reuse the AMMA platform set up for our framework. Diagnostic is a model containing the outcome of the verification process; it will conform to a self-described meta-model which is represented in Figure 20.b.

The benefits of this approach are: (i) the Verifier component is generic (i.e. it is always the same for every property to check) and thus it is maintainable. The properties are expressed through ad-hoc models and they are “engineerable”, i.e. many software engineering techniques (e.g. automatic generation of the properties to verify)
may be applied to the properties in an MDE fashion. The most relevant drawback of this approach is that a meta-
model for the properties’ models must be designed. This is not straightforward because it has to cope with different
kinds of properties and because it must be complete in terms of power of expression (i.e. it is not acceptable that
a specific property cannot be checked only because the meta-model does not contain the constructs to express it).

Transformation-based approach (Figure 21). This approach was presented and developed in [13].

![Fig. 21. Transformation-based approach for maintaining models’ consistency.](image)

The input model is only one, i.e. the model being validated. The properties to verify are embedded into conditional
OCL-based expressions that guard the rules of an ATL transformation. The Diagnostic is expressed through the
same meta-model described in the previous approach. This approach does not “constrain” the properties into models,
thus we do not need to develop a meta-model for properties. Possible drawbacks are that for each set of properties
(or even for each single property) the architect has to develop an ATL transformation that embeds them. Moreover
modifying a property implies the corresponding modification to the transformation that verifies it: the architect is
forced to know the ATL language and tracing problems between the specification of the properties and the Verifier
component may rise. Future advancements of DUALLy may include such a mechanism in order to verify the stated
quality needs.

(ii) Extendibility and interoperability among ADLs and SA UML profiles: within DUALLy each of the
members of the standard OMG technical space, namely model instance (M0), model (M1), meta-model (M2)
and meta-meta-model (M3) are fully extendable and fully supported within DUALLy by means of profiling and
meta-modeling (i.e. heavy-weight profiling). Each of the layers of the DUALLy architecture as well as all the
DUALLyzed (i.e. DUALLy integrated) technologies can be customized and tuned for any purpose. Moving on
to integration, DUALLy presents mechanisms to integrate ADLs and UML-SA at each step of the way; as an
additional pro, DUALLy does not hard code UML within its infrastructure, but rather retrieves it dynamically
from its environment, proving also version independent and therefore quickly able to adopt and adapt future UML
innovations.

For what concerns interoperability we report our experiment of DUALLyization of AADL which is, as depicted
in Section V, a complex and more specific ADL with respect to both Darwin and ACME. Upon this experiment
we will argue about two points: the first being how each ADL can be DUALLyzed while the second being how
even most different ADLs can be successfully related with each other through A0.

Exploring the first point, we show how the elements of A0 can be combined in order to semantically represent
the artifacts of AADL. For instance, AADL provides different kinds of components which can be specified at both
software and hardware level (e.g. system, process, thread, memory, bus, etc). $A_0$ contains only a way to specify a component, $\textit{SAComponent}$, but this artifact can be coupled with the $\textit{SAType}$ artifact so as to reproduce the different kinds of AADL components. The same approach can be applied to AADL ports that can be of three different types, data, event and data/event, singularly reproduced through the $\textit{SAPort}$ element combined with different $\textit{SAType}$ artifacts. A different situation is encountered when dealing with the “flow” constructs, used in AADL to specify information flows in the overall SA. In fact, this particular construct is mapped in the $\textit{SABehavior}$ package through activity diagrams that $\textit{DUALLY}$ inherits directly from the UML meta-model.

Let us now discuss the second point. As we already stated, AADL can be considered a more specific ADL with respect to ACME and Darwin, in the sense that it provides a number of specificities that cannot be directly mapped to elements of ACME or Darwin (and its behavioral counterpart LTSA). Therefore, it becomes interesting to discuss the effect of both passages, namely, from Darwin (or ACME alike) to AADL and viceversa, obviously through the $A_0$ passageway. In the first passage we are going from a more generic technology to a more specific one and this implies that each element present in the source technology will be directly reproduced in the target technology. Additional transformation rules might be inferred via a thorough semantic analysis of constructs of the more specific ADL. In AADL, for instance, the only possible way to express non deterministic behavior is to use, within a $\textit{thread}$, a sequence of $\textit{subprogram}$ calls [24]. Therefore, a transformation rule will specify that Darwin/LTSA components with associated behavior will become AADL threads and successively that the associated (LTSA) state machine will generate an appropriate sequence of subprogram calls to be associated to the threads themselves.

In the second passage, specificities of AADL are reflected to $A_0$ thanks to the AADL $\textit{DUALLY}$zation as explained above. However these specificities cannot be reflected to Darwin and thus, when going from $A_0$ to Darwin and back, we may loose information. To solve this problem it is possible to provide a mechanism to safekeep this information by storing it as additional information. More details are provided in Section VII.

(iii) Masking the complexity: the process suggested by $\textit{DUALLY}$ is organized in two different phases: typically the first phase (that ends with the transformations generation) is executed only once for each couple of meta-models/profiles (i.e. ADL meta-model/profile and $A_0$ or two meta-model/profile in the case of a direct link) and it is managed by a meta-modeler who retains a deep knowledge of the meta-models to weave; the second phase is executed every time a software architect needs to pass from a model source to the corresponding target model. $\textit{DUALLY}$ works at two abstraction levels, therefore masking to the software architects the model transformations technology. The role played by the software architect is in fact that of a final user that has only to model the system SA on an ADL and he automatically will have the system SA in each $\textit{DUALLY}$zed ADL.

(iv) Integration with MDA technologies: since $\textit{DUALLY}$ is built around AMMA, which is an eclipse project, it could be easily integrated with MDA technologies already available within the eclipse community (e.g. Business Process Modeling Notation (BPMN), Eclipse Model-to-Model Transformation (M2M), Model To Text (M2T),
VII. CONCLUSIONS AND FUTURE WORK

In this paper we presented DUALLY, an automated framework that allows architectural languages and tools interoperability through model transformation technologies. DUALLY’s goal and rational is to solve the problem of having a proliferation of ADLs and UML notations for SAs not supported by effective interoperability. DUALLY brings together different architectural languages through a common semantic core, called A0, that provides the infrastructure upon which to construct semantic relations among the different ADLs. DUALLY is implemented as an Eclipse plugin.

In order to fully automate DUALLY’s infrastructure a set of mechanisms for the (semi)automatic generation of the Weaving Model (WM) will be provided. This is possible thanks to the so called matching transformations [21]. Such transformations select a set of elements from the input meta-models and produce links between them; these links are then captured within a weaving model. Matching transformations create these weaving models by executing matching heuristics guided by similarity principles. This value can be calculated by using, for example, the edit distance between the names of the elements of the input meta-models, equality parameters between attributes or classes or by using the structural relationships (e.g. containment) between model elements. An important aspect is that matching transformations may be applied in sequence, thus creating a chain of transformations that progressively forge the definitive weaving model.

Another interesting future direction is engineering and developing a mechanism to compute and track the upgrades or adjustments made on a target model (generated by the DUALLY transformation network) so as to reflect those changes within the source model in a round-trip fashion. This mechanism will be engineered to calculate any changes made through model differencing technologies. This mechanism must also consider that some ADLs might not be able to represent all the possible information contained in A0. The unrepresentable information has to be safekept in a special package containing an entry for each A0 element that has additional information to be kept. This entry will be retrieved and applied to the model when going back to A0; retrieval will reflect possible modifications made on the overall architecture within the ADL we are coming from.

Another interesting pathway is that of exploring how two ADLs may be interchanged both directly (using a single weaving model from source to target ADLs) and through A0 (using a weaving model from the source ADL to A0 and another weaving model from A0 to the target ADL). A very important issue is how to manage the possibility of having both direct transformations and transformations through A0 between two different meta-models (ADLs). More specifically, the problem is twofold: there will be two models obtained from the two kinds of link and these will likely be different, the user, on the contrary, would expect as output a single model. This problem can be tackled at two modeling levels:

6BPMN (eclipse.org/stp/bpmn), M2M (eclipse.org/proposals/m2m), M2T (eclipse.org/modeling/m2t), Birt(eclipse.org/proposals/eclipse-birt), Omondo(eclipsedownload.com), AndroMDA(andromda.org).
1. **Meta-model-level solution**: ATL supports the possibility of having a transformation among \( n \) models, thus we can craft a particular kind of HOT that takes as input the three weaving models and the corresponding meta-models and produces the final ATL transformation that reflects the “union” of the logic of all the weaving models.

2. **Model-level solution**: for each couple of meta-models we can develop a merging transformation that takes as input the two models and merges them in some consistent way.

We are currently investigating on the benefits and drawbacks of the two solutions (or the existence of alternative ones).

Further effort could also be invested in considering DULLY as base infrastructure upon which to construct a “third generation” ADL which can be defined as a customizable and extendable ADL so as to satisfy system-specific stakeholder concerns. In fact, in case we are interested in modeling heterogeneous characteristics that are not contained in one specific existing architectural language, the idea is to create a new ADL as result of an extension of our own “third generation” ADL (\( A_0 \) augmented with previous extensions) by merging it with characteristics specific to other existing architectural languages. This need raises from the goal of evidencing inter-relations between different analysis and modeling techniques and in particular between functional and non-functional aspects that would not necessarily emerge from distinct analyses. The challenge is to automatically build the extension while keeping firmly the semantics and characteristics of the original architectural languages. It is important to note that in case of different extension needs, the software architect will be provided with facilities to cut off unwanted features while finely tuning his own instance of “third generation” ADL. This possibility alone, clearly presents itself as a key difference with respect to current extensible architectural languages. A first rough experiment in this direction is reported in [43]: weaving operators are used to extend profiles/meta-models in a conservative way: deletions of constructs are denied while allowing for their specialization or refinement exclusively.

Two operators were devised:

(i) the *inherit operator* projects within a profile/meta-model, elements of other profiles/meta-models which are linked through an inherit stereotyped association. As a result, elements already existing within the profile/meta-model to be extended, are augmented with data coming from the elements to which they are related through inheritance.

(ii) the *integrate operator* extends profiles/meta-models with entire blocks of data coming from other existing architectural languages which provide constructs not present already.

In [43] we showed the extension of \( A_0 \) with fault tolerance information with the aim of capturing in architectural components both normal and exceptional services. The exceptional part implements the responses of the component to exceptional situations, by means of exception-handling techniques.

Concluding, an ultimate future research direction is to reuse the DULLY framework in contexts other than the architectural languages context. This implies the definition of a new \( A_0 \), customized for the new context, and thus reusing the DULLYzation technology to bind the different approaches together as required by the new context itself.
ACKNOWLEDGMENT

The work is also partially supported by ARTDECO (Adaptive inFRasTructure for DECentralized Organizations), an Italian FIRB (Fondo per gli Investimenti della Ricerca di Base) 2005-2009 Project. The authors would like to thank Davide Di Ruscio and Nenad Medvidovic for their valuable support and comments.

REFERENCES


March 31, 2008


List of Acronyms:

AMMA : ATLAS Model Management Architecture;
AM3 : ATLAS MegaModel Management;
AMW : ATLAS Model Weaver;
ATL : ATLAS Transformation Language;
ATLAS : Team at INRIA institute (Nantes, France);
ATP : ATLAS Technical Projects;
BT : Basic Transformations;
CTS : Compatible Time Sharing;
DSL : Domain Specific Language;
Ecore : Eclipse implementation of MOF compliant meta-meta-model;
EMF : Eclipse Modeling Framework;
HOT : Higher-Order Transformations;
IECS : Integrated Environment for Communication on Ship;
IECS-MS : Integrated Environment for Communication on Ship Manager-Agent;
KM3 : Kernel Meta-Meta-Model;
MDA : Model Driven Architecture;
MDE : Model Driven Engineering;
MMx : generic meta-model;
Mx : generic model;
MOF : Meta Object Facility - OMG official Meta-Meta-Model;
OMG : Object Management Group;
UMF : Multi Functional Unit (the acronym comes from the Italian “Unitá Multi Funzionale”);
UML : Unified Modeling Language;
UML2 : implementation of the UML meta-model for the Eclipse platform;
UML 2.0 : version 2.0 of UML;
WM : Weaving Model;
WMM : Weaving Meta-Model;
XMI : XML Meta Data Interchange;
XTEAM : an MDE focused extension to the xADL framework.