RIGOUROUS ENGINEERING OF SOFTWARE ARCHITECTURES:
INTEGRATING ADLS, UML AND DEVELOPMENT METHODOLOGIES

Nicolas Guelfi, Gilles Perrouin,
Luxembourg University of Applied Sciences
Software Engineering Competence Center
6, Rue Coudenhove-Kalergi, L- 1359 Luxembourg-Kirchberg, LUXEMBOURG
{nicholas.guelfi, gilles.perrouin}@ist.lu

ABSTRACT

Architecture engineering is an area that is under construction for being transferred to software engineers. Many results exist from high-level formal and conceptual approaches to basic good practices taken from industrial experiences. Descriptions and methodologies should be at least considered for architecture engineering. Two interesting approaches to architecture description are available: Architecture Description Languages (ADL) or modeling languages (especially UML). In the first case, efforts have been made to provide rigorous integrated languages dedicated to architecture engineering but unfortunately these languages are incomplete and not ready for technology transfer. Concerning UML, many approaches use UML models for partial architecture descriptions. In both cases, a few efforts have been made on methodology. This paper presents the current state of the art covering architecture descriptions and development methodologies for architecture engineering. We present a first integrated approach, called FIDJI that tries to take advantage from ADLs, UML, and some development methodologies. FIDJI uses UML for its models, MDA for its layering and ADLs for precision. Concerning methodology, FIDJI follows the HP recommendations and the notion of refinement and contracts taken from formal development methodologies. Finally, we briefly illustrate the use of the FIDJI approach by means of a real web-based distributed application.

KEYWORDS

Software Engineering, Software Architectures, ADLs, UML, Development Methodologies.

1. INTRODUCTION

Although the concept of “Software Architecture” was first described in the 60s, a significant interest among the research community is only 15 years old, and the industrial community’s interest is very recent. The reason for such a late interest is because in the early 90s most of the information systems were concentrated on single machines, and others were distributed on only a few machines. Things began to change in the mid 90s, with the emergence of low cost, fast networks making distributed systems more reliable and flexible. The current trends in pervasive computing and devices diversity (PDA, phones…) will make the architecture of systems (the “big picture”) much more complex. Therefore we need better architectural descriptions; this is the main issue addressed by this paper.

Software Architecture’s research results are widely ignored by the industrial community for which a good architect is mainly an experienced person. Currently the most used technique for distributed systems engineering is based on “best practices” documents that do not necessarily include precise models and descriptions.

We consider that, to be useful, architecture engineering should focus on convenient tools for architecture description and limpid associated methodologies. On one hand, architecture description languages (ADLs) [1] address more the basic concepts of system architecture such as components and interfaces than the global design models for architecture engineering. On the other hand modeling languages (essentially UML like languages) have difficulties to adapt their meta-models to architecture specific ones. In both cases, we see a major lack on the methodological aspects. There are few development methodologies covering architecture engineering. Some interesting initiatives that are going towards this objective of development methodologies covering architecture engineering are RUP, MDA [2], and HP recommendations initially started in the Fusion 2.0 project [3].

The FIDJI project [4], in which this work takes place, studies tools and methods able to enhance rigor in software design. Inspired by a strong formal methods back-

1 This work is supported by the Luxembourg Ministry of Higher Education and Research under the Title 1 project n°MEN/IST/01/001

2 We use the MDA definition of component “individual unit of computation” that can be managed at different abstraction levels.
ground [5], this paper focuses on architecture centric development processes and on existing technologies improving the correctness and the accuracy of software architecture description. It also concentrates on several ways to improve software development: frameworks [6] and components. Moreover, it provides methods for their design (refinement, patterns based transformations) [7].

The problem addressed in this paper is the integration of industry best practices, architecture description languages, and UML in a rigorous and methodological approach. In this context, we propose an integrated solution, which is illustrated through a real, medium-sized case study.

In the first part of this paper we examine the existing solutions for architectural description, in the second part; we highlight the possible methodologies that include architectural engineering. The third part presents the basic concepts of a possible integration of ADL, UML and development methodologies. Finally, in the last part we illustrate our approach with a medium-sized case study.

2. STATE OF THE ART FOR ARCHITECTURAL DESCRIPTION

2.1 Industrial Viewpoint

Since software products are built, developers (the term “architect” has been only recently recognized by the business community [8]) try to describe their applications. Because specific notations for describing architecture aspects were missing for many years, a “boxes and lines” culture has been adopted by the development community widespread among people. Architectural description was empirical (and in a lesser extent is still yet) and everyone had its own recipes to integrate functional requirements. Unfortunately non-functional requirements are hardly ever taken into account. Recipes are better when experience flavors the dish. Good descriptions result from years of failures, good integration between technical constraints, quality constraints and enterprise culture. In the last five years, use of standardized notations and guidelines for architecture description has been progressively introduced in development projects. But the maturity level is not yet reached due to the lack of rigor and insufficient methodological approach.

Our expertise over the last 14 years with industrial partners, either through R&D projects or professional organizations (like the SPIRAL network in Europe), has shown that current practices in system and software architecture engineering rarely reach the level 2 (“Repeatable”) of the Capability Maturity Model of the SEI.

2.2 R&D Viewpoint

On the other side, the R&D community designed several formalisms - integrated or not - in software development methodologies. This research effort began in the 90s when systems became distributed and complex enough such that empirical approaches dramatically increased risks and threatened successful termination of projects. Bigger projects implied having more people (not only technical ones) working on them making proprietary recipes hard to share. The results of this effort led to different technologies:

- Notations and Documentation templates [8,9]
- Architecture Description Languages [1,10]

Several Object Oriented notations used in development methodologies have been developed in the beginning of the 60s (OMT, BOOCH). Fusion [12] is a first attempt at reconciling those ones and proposes a complete notation; architectural concepts [13] were not well defined at this time. Concerning notations industry as well as the academic world has recently converged to OMG’s UML. This modeling language has standardized the drawing of boxes ("classes") and lines ("associations"), making CASE tools popularization possible. However, UML is not the panacea for the following reasons:

- The semantics are not well defined. In order to be expressive and convenient, UML defines many useful types, but their interpretation may differ among stakeholders and confuse them.
- The graphical notation used is sometimes too cumbersome to describe complex systems. But this point can be solved using views (such as the 4+1 view model [14]). Stereotypes [15] may also help to give a more significant representation of a system.

There are proposals to integrate architectural concepts either in UML directly [16] or to extend it [17], [18]. Moreover, templates for software requirement documentation were developed. The most “standard” one is the Software Requirements Specification from IEEE [19], other templates can be found in the industry [9,3].

ADLs, issued from research, are one solution to the specific architecture description issues. They provide a scientific basis to the architectural description by defining only a few concepts and provide ways to construct (architectural styles; reusable patterns for architectures), describe, and evaluate architectures. A comparison of these languages can be found in [1].

Architecture Oriented Programming is a new paradigm to develop software systems. By using an architectural development environment, it is possible to generate skeletons of classes representing architectures [11], to some extent such architectures are considered as “executable”.

---

3 We conform to the IEEE p1471 recommended practice, in which a system is “a collection of components organized to accomplish a specific function or set of functions”
3. DEVELOPMENT METHODS FOR SOFTWARE ARCHITECTURE ENGINEERING

Having a clear, precise, and standardized notation for software architecture is not sufficient. If not used within a convenient methodology, projects are unlikely to be successful upon completion. We think it is important to initiate methodological ideas in order to address this complexity with the maximum of rigor and to organize the development around architectural aspects. We studied existing methodologies searching where exactly software architecture is placed. In the following paragraph is presented the result of this study.

In the Rational Unified Process [20]: Software architecture is involved in the inception and the elaboration phases (the four phases of RUP are inception, elaboration, construction, transition). In the inception phase, architecture possibilities are evaluated with respect to scenarios. In the elaboration phase the main architectural job is processed. Components (made?/bought?/reused?), are selected and evaluated according to the component policies defined before. This work on components can provide feedback to reconsider or not the whole architecture. Since numerous iterations can be scheduled in these phases, the architecture can be modified until it becomes stable. (This asset is not specific to RUP, but to every iterative method).

Another methodological tool sponsored by the Rational Corporation is the Reusable Asset Specification [21]. This initiative aims at standardizing software reuse documentation. The UML notation is used to describe the specification; the terminology used is the same as in RUP. In order to reuse architectural elements in the architecture engineering process, these elements must be specified using assets. An asset is divided into three parts:

- **Logical asset structure**: this part is described itself in the following subparts: Overview, Usage, Classification (describes the context or domain where this asset should be used), and Solution
- **Asset relationships**: describes the relationships within the asset
- **Physical asset structure**: Describe how the asset is described physically; XSD files, source code...

Core elements can be extended to describe a specific set of assets. This extension is realized via MS Word templates and XML schemas.

The Reusable Asset Specification is still under development and valuable information can be found here [12].

In the OMG Model Driven Architecture (MDA) [2] (that is based on the Catalysis [22] approach for some concepts) the architecture is at the center of software development. The key concepts are:

- **PIM**: Platform Independent Model, an abstract description of the architecture describing what the components are doing but independently of any implementation details (for instance SOAP-based communication between two web-services should not be specified).
- **PSM**: Platform Specific Model, the architecture is “instanted” here on a concrete technology such as Sun’s J2EE or Microsoft’s .NET.
- To navigate between models the zooming in/out mechanism of catalysis is also employed: In fact abstraction helps understanding from coarse-grained components to fine-grained ones.

This approach maximizes the work on architectures since it remains usable (the PSM can be used in many projects sharing close requirements), but the mappings between PIM and PSM are not that simple even if UML technologies such as profiles [15] can help to automate the mapping. Multiple views and viewpoints accordingly to the IEEE p1471 are also encouraged to capture the whole architecture.

Concerning the Fusion development methodology, architectural phase was not covered in the Fusion 1.0 method. **Fusion 2.0** [3] has corrected this. Unlike RUP in which architectural development is distributed in the two first phases and seen as a discipline, Fusion 2.0 defines an architecture phase placed between analysis and design. Architectures in Fusion 2.0 are split in “conceptual architectures” and “logical architectures”; the first one is more informal and abstract, when the second one is very pre-
The architecture development process is divided in six steps as follows:

- **Architectural styles review and selection:** A study must be made to identify architectural styles relevant to the domain of concern. Requirements phase provides criteria to select them. Then, a new style can be built as composition of others.

- **Informal design of the architecture:** this step must identify the components and their responsibilities and focus on invariants of the system. Risks should also be evaluated.

- **Development of the conceptual architecture:** This step focuses especially on obtaining precise components collaboration where the risks are the highest. Naturally it is not done in one day, and this architecture may be refined as many times as necessary.

- **Development of the logical architecture:** This step describes in a very detailed manner the collaboration between components and their interfaces (depending on the size and distribution the description may vary in accuracy).

- **Rationalization:** Once the architecture is designed, architects and developers have to check whether it satisfies non-functional requirements or not, and overall consistency of the architecture (pre/post conditions…).

- **Design guidelines:** This is a set of rules to be followed by designers. This could cover security policies, communications, mechanisms etc…

### 4. THE FIDJI APPROACH TO SOFTWARE ARCHITECTURE ENGINEERING

The intent of the FIDJI project is to define a methodology for distributed applications in JAVA (mainly but not only) based on a scientific approach: In software intensive systems the more rigorous we are, the less chance we have to fail (for further information on the FIDJI project, see [4]).

We also wanted to propose a pragmatic approach (the key concepts are summarized in figure 5.1). For instance, specific ADLs notations limit their democratization in a world where UML is the standard for software description.

This approach identifies the following steps or disciplines that are inspired from Fusion 2.0: **Requirements, Domain analysis, Architectural, and Design.**

#### 4.1 Requirements step

This step is very classical and similar to many other OO methods. This is a set of use-cases, both graphical and textual. Since our approach is component-based the emphasis should be put on inter use-cases relationships. Non-Functional Requirements (NFR) such as quality policies should also be considered during this step. During this step, we may also define _contracts_ [23]: contracts are statements that have to be satisfied by the final system. For example you may specify for a number adding component that it would return the correct sum. Contracts are one tool to achieve rigor and may be very useful during component choices…

#### 4.2 Domain Analysis step

Domain analysis is gaining interest as applications go further into customers’ businesses. Another interest is the fact that many APIs and components are available today, and domain analysis provides criteria to choose them.

#### 4.3 Architectural Step

In our opinion, this step is critical. The aim is to develop a flexible and efficient architecture that has the capacity to evolve over the years.

##### 4.3.1 Abstract Architecture

The first sub-step is to choose the architectural styles that will be used in the application. From domain analysis and requirements we should be able to identify these styles (and develop specific ones) that will act as guides for the remaining design. As one style addresses one kind of problems, it is documented as an asset following the RAS specification. In this step there is also work on components: You can define a component policy to have criteria to handle components and domain analysis can be also helpful. At the end of this process you have an informal architecture where components are loosely related. To eliminate _a priori_ non-viable relationships small feasibility prototypes can be developed. This step corresponds to conceptual architecture in Fusion 2.0 and, assuming that components are technology-independent, to the PIM in MDA.

##### 4.3.2 Concrete Architecture

The main objective of this step is the concrete and precise description of the architecture. It is an important step for two reasons:

- This step is useful to share the vision of the architecture among stakeholders to ensure that everyone (architects, developers, businessmen, customers) agrees with it
- It provides the best support to any evolution or change in the architecture. The better the description is the less chances we have to violate the architecture’s vision and thus encounter problems. This is also a good protection against employees’ turn-over since the “memory” of the architecture is kept.

In the ADLs we find all the concepts we need to construct a precise and consistent description of the architecture.
Some of them even provide checkers and others tools to ensure architecture viability or to design small prototypes. However, these languages - not recognized by the industry - do not have convenient tools (the intent of their authors was to illustrate fundamental notions rather than provide real tool support). On the other side, UML can be improved to better support architecture description (for example, architectural styles are not easy to represent). To address this problem here are presented a few solutions:

- **Modify the UML directly:** It is possible to modify the UML core using MOF (Meta Object Facility) and make it able to support architectural concepts [18]. This is the cleanest solution, but it generates an “incompatible UML” that will suffer the same drawbacks as other ADLs (especially tool support).
- **Extend and constrain the use of UML:** The latest specification of UML (1.4) proposes a profile mechanism allowing us to extend the UML to add semantics and rules (in OCL for instance). Together with methodological recommendations, existing CASE tool support can be kept as well as good architecture descriptions. [26]. This approach can be improved by the use of a formal textual notation (in addition of OCL) in complement of diagrams; An XML based ADL such as Xadl [10] brings the rigor necessary to architectural descriptions.
- **UML 2.0 is currently under development:** Some propositions (RFP) [27] include the support of architectural concepts. But at this time we do not know whether they will be included or not.

One of the most difficult issues in this step is to provide a concrete architecture that support the functional requirements and which validates its associated non-functional requirements. Architectural frameworks [24] can be a partial solution that must be integrated in this step. They provide design diagrams as well as source code that can be directly reused. Naturally, to fit particular projects they have to be extended; this is called framework specialization. To illustrate this idea, in the FIDJI project we have worked with the Rational Company on a J2EE based Architectural Framework for an online auction house (an auction house is a system able to identify sellers, buyers and record transactions), which is called Pearl Circle [25] and its improvements based on patterns JAFAR [6,7]. Using and specializing a Framework implies that the target platform is already chosen, thus it is equivalent to the PSM in the MDA. Obviously, developers do not have to conceive - a framework for each application, - they construct theirs by leveraging existing ones, and you make them evolve as the successive domain analyses suggest it. Thus the concrete architecture step leads to a carefully written description of the architecture and its software implementation.

### 4.4 Design step

Once architecture is defined, developed, and stabilized, design of the other services of the application is possible. The main concept to take into account here is *Refinement*; at first services are coarsely defined and then iteratively detailed using views at different levels. This concept is also described as Zooming in/out in MDA.

In many methods this concept is present. But if everybody agrees on the fact that having multiple levels of abstraction allows a complete view of the system or small very detailed one, nobody seems to provide methods to effectively implement this refinement. The reason for this is quite simple there is no “ready to use” solution, since it concerns the way architects works and this is very hard to automate. However some tools can help people in these tasks: Architectural Styles and Design Patterns are often accompanied with diagrams or code, so that using of them can reduce the number of services to be refined (since they already are!). Some extension mechanisms of UML can also be useful.

### 5. EXPERIMENTATION ON THE LUXDEAL CASE STUDY

As mentioned above, one part of the FIDJI team has worked on an e-commerce Framework (JAFAR). The Luxdeal case study is very close to an auction application with some differences; the system focuses on exchange: for instance you exchange one hour of JAVA course against one hour to learn how to play piano. To illustrate the benefits of using Frameworks in software development, we carried out an experiment: “realize a prototype of Luxdeal in 15 days”. Three teams composed of two people worked on it; one team with the Framework, another team without (The third team - who have developed the Framework - was devoted to support, organize and coordinate of the experiment).

Gilles Perrouin was involved in the team who was using the framework and used this opportunity to try these ideas on a small concrete software development. Problems encountered by the teams have helped us to improve the framework documentation, and figure out some key issues and benefits of our approach.

However, the conditions of the experiment (designed for other purposes), did not allow us to experiment the whole FIDJI development process. Requirements of the prototype were provided and the JAFAR framework brings its own architecture on which software is realized. It means that the abstract and concrete steps were already done using informal documentation, UML models, and framework documentation.
The development process within the JAFAR is as follows (see Figure 5.1):

- **Creation of Use-Cases**: This step is found in all development processes. It helps to formalize and understand the requirements. It has a supplementary asset in an architectural framework: it helps to find whether some elements are present in the framework or not. It is mandatory to make a distinction between developer entities and framework entities since the resulting code is a merge of the code of the framework and developer code, same entities could lead to conflicts and lack of consistency in the final application.
- **Creation of key abstractions**: Key abstractions (this terminology is provided by Rational) take as inputs the results of domain analysis i.e. creation of business entities and their properties. For instance in an auctioneering application, the entities “Offer” and “Item” appeared clearly in domain analysis. These key abstractions are mapped into Entity beans by a pattern provided by the framework.
- **Creation of business packages**: This corresponds to the business logic of the application. Described as UML subsystems (the closest UML element to a component), they are automatically transformed into Session Beans by a pattern provided with the framework.
- **Creation of presentation layer**: Once the business logic is defined, we begin the development of user interface (in luxdeal application it was JSP pages) pages are created and their logic is defined in dispatchers (classes implementing the navigability and the logic of pages, one of SUN J2EE’s patterns [28]).

The results of the experiment were interesting: we obtained a much more reliable and scalable J2EE solution than the other group without the framework (who was using just Servlets and JSPs). Such a development (using EJBs) would have not led to a complete application in only 15 days without JAFAR.

However, aside from technical problems (due to the youth age of the Rational XDE tool), a clear vision of the overall methodology to use and to specialize the framework may help. During the development, we were sometimes confused about the next step to accomplish. A “to do list” principle as suggested in Fred [11] could bring some very useful information. Another point to cope with is the fact that in the current version of the framework, it is not possible to re-apply patterns. This means that for now the development process corresponds to the waterfall process model.

### 6. CONCLUSION

After having presented the state of the art of software architectures engineering, measured the gap between industry and research in this domain, we proposed our approach that is standing on the border line of the two worlds: Industry and Research. We also showed significant results of the FIDJI approach. However, our approach has to be improved in the following directions:

- **Architectural Description**: We identified an interesting compromise to describe architectures with UML, but we also have to make it supported by tools and provide methodological assistance to this fastidious task.
- **Frameworks**: JAFAR framework needs to be improved (technically and methodologically) in order to show the benefits of this technology.
- **Apply the overall FIDJI development process on the LuxDeal case study based on the JAFAR framework.**

### 7. ACKNOWLEDGMENTS

We would like to thank Wojtek Kozaczynski and Jim Thario who gave us a lot of inputs on all the subjects covered by this paper. We also would like to thank the FIDJI team members who were of great help for this work.
8. REFERENCES


[22] Desmond F. D’Souza, Alan Cameron Wills, Objects, Components, and Frameworks with UML the catalysis approach, Addison-Wesley 1998.


[27] OMG, Request For Proposal:UML 2.0 Infrastructure RFP, September 2000