Goals Model-Driving Software Architecture*

Elena Navarro  
Department of Computer Science, UCLM, Albacete, Spain  
enavarro@info-ab.uclm.es

Isidro Ramos & Jennifer Pérez  
Department of Information Systems and Computation, UPV, Valencia, Spain  
{iramos |jeperez}@dsic.upv.es

Abstract  
On every new development new issues are taken into account to cope with either environmental or the stakeholders' needs that are evolving over time. Several approaches have faced this problem. Some of them exhibit this evolution ability by using static design/compile-time techniques whereas others introduce this ability in the system at run-time. Nevertheless, in both cases evolving requirements give rise to the need of adaptability which is inherent to every software development. This paper sketches our work in this field in which we are concerned about the ATRIUM methodology to guide the reflexive development of architectures from the software requirements. In particular, we are detailing the first step of this methodology, i.e., the definition of the goals model whose constituents are the fundamental basis for the overall process defined in ATRIUM proving its suitability for obtaining traceable architectural models. It provides our work either to its ability to specify and manage positive and negative interactions among goals or to its capability to trace low-level details back to high-level concerns.

1. Introduction  
Dynamism and evolution are currently two main software concerns. On every new development new issues appear related to the customisation of the software. They attempt to accomplish either environmental or the stakeholders' needs that are evolving over time. How to overcome deficiencies and limits exhibited by traditional development methodologies is a clear challenge that has been faced by several techniques. Some of them try to introduce this evolution ability by using static design/compile-time techniques whereas others introduce this ability in a way that evolution appears at run-time.

Aspect Oriented software Development (AOSD) [1] is related to the first type of techniques providing advantages in expressiveness by the separation of concerns. Both functional and non-functional needs, such as performance or compatibility, of the system's behaviour can be separately acquired and specified across the development lifecycle. From requirements [24] to architecture [23], design [27] and implementation [13], each concern has to be traced and easily adapted to new stakeholders' needs.

Concerning the second category of techniques, existing approaches provide high-level architectural descriptions that are well suited for runtime evolution. Architectural reflection [28] is a clear example in this sense, offering facilities for dynamic reconfiguration, in terms of composition and/or connection, to existing systems without modifying them. Several approaches [19, 23] make use of a meta-level in such a way that software systems are aware of their architectural properties and are able to adapt themselves at runtime in a simple way.

In both cases, however, evolving stakeholder's requirements give rise to the need of adaptability. It is inherent to every software development that expectations about the system are evolving over time. The Requirements Engineering process (RE) establishes the foundation on which the system-to-be should be implemented. Therefore, it has to be able to identify and define this need of adaptability into its artefacts in such a way that they can be traced to low level abstraction artefacts.

A methodology ATRIUM (Architecture generaTed from RequIrements applying a Unified Methodology) [20] illustrates a process to concurrently define requirements and architectural artefacts, in the same way that Nuseibeh proposes [22] to obtain architectural artefacts that provide ability for dynamism and evolution by means of AOSD and reflexive techniques. In this work, we are detailing the first step of this methodology, i.e., the definition of the goals model whose constituents are the fundamental basis for the overall process defined in ATRIUM.

This work is organized: Section 2 introduces ATRIUM methodology. Section 3 defines the goals model. Section 4 presents a case study. Related work is presented in Section 5. Conclusions round up the paper in Section 6.

2. The ATRIUM Methodology  
This methodology is concerned with the definition of software architectures from functional (FR) and non-

---

* This work has been funded by the Spanish CICYT project DYNAMICA (TIC2003-07776-C02-02)
functionality of software, we define functional requirements (NFR). With this aim, it provides
the analyst with a guidance along the process from an initial set of requirements to an architectural instance.
ATRIUM is a not a product oriented approach but a process oriented one. It does not describe a set of metrics
to determine if the requirements, functional and non-functional, are implemented into the final software. On the contrary, requirements and architectural design decisions are defined concurrently in such a way that every decision is made to achieve a given requirement.

### Step 1.- Goals model (GM) definition

In this step, the set of goals to be accomplished by the software system are defined by means of both functional and non-functional requirements. An informal set of requirements, stated in natural language, is the input to trigger the model definition (see section 3). The elaboration of the Goals and Scenarios Models (see step 2) are two intertwined processes. The GM is operationalized at the next step by means of the Scenarios Model (SM). Furthermore, the analysis information provided by the SM helps us to refine and identify new goals at the next iteration. Therefore, both models are coupled, as in [25], with a meaningful advantage in terms of traceability.

### Step 2.- Scenarios model (SM) definition

The analyst has to identify the set of scenarios which operationalize the established goals and compose them to form an iterating and branching model of the system’s behaviour. Each scenario depicts the elements that interact to satisfy a specific goal and their level of responsibility in achieving a given task. These elements are shallow-components, i.e., a rough description of the components that appear into the final software architecture. Use Cases(UC) and Message Sequence Charts (MSC) [29] are employed for the construction of the SM. The former provide us with a visual metaphor notation for scenarios, where the identified shallow components are the involved actors. Nevertheless, when we speak about Software Architecture[7], we have to identify not only the involved elements but also the relationships between them. MSCs identify the coordination structure through the temporal sequences of interaction events. Several alternative scenarios can operationalize the same goal, just like several alternative programs can implement the same specification. In order to offer the best approach, they have to be analysed caring about conflicts that may arise among the operationalized goals.

### Step 3.- Collaborations definition

The third step is aware of the collaborations among the identified shallow components. The collaborations realize the UCs through collaboration diagrams. The objective is to obtain a proto-architecture for the concrete system by using related architectural styles [26], interaction patterns and the architect’s ability. Additionally, the connectors are defined according to the components interactions. These first class citizens are required to achieve a loose coupling between the components.

### Step 4.- Formalization

A semantic check and analysis of the Models and the proto-architecture is required to identify eventual conflicts, e.g., different scenarios resulting in incompatible architectural configurations, and obtain the best alternative to avoid (or minimize) them. The artefacts, defined in the previous steps, are specified in the 3APL language[10], whose interpreter helps us to validate them. A set of derivation rules are provided to generate a formal specification from scenarios, goals and collaborations, in order to assist and speed up the formalization process. This specification is validated and then used for an automatic compilation process in the next step.

### Step 5.- Compilation

Using the formal model and a set of generation patterns, the translation from the requirements model to an instantiated PRISMA model specification is accomplished. PRISMA [23] (PlatafoRma oasIs para Modelos Arquitectónicos, OasIs Platform for Architectural Models) has been selected as the reference model to specify architectural elements by integrating Component-Based Software Development (CBSD) [30] and AOSD [1] approaches. This architectural model is able to be compiled into a concrete target system preserving its reflexive properties. This step is assisted by the engineer in order to refine the architectural elements.

As ATRIUM is intended to be iterative and incremental, a feedback is provided from Step 5 to 1. In this way, all the models are up-to-date all over the process.
3. Goals Model: An Artefact for RE

In the context of Requirements Engineering, the Goal-Driven Requirements Engineering paradigm [15] has proven useful to elicit and define requirements. More traditional systems analysis tools, such as UCs, focus on establishing the features (i.e. activities and entities) that a system will support. Nevertheless, Goal-based approaches [5, 6] focus on why systems are constructed providing the motivation and rationale to justify software requirements. They are not only useful for analyzing goals but also for elaborating and refining them.

Additionally, we have to bear in mind that architectural models are a bridge between requirements and the system-to-be [7] providing us with a lower abstraction level. They are used as intermediate artefacts to analyse whether the requirements are met or not. Therefore, this paradigm has two advantages that make it appropriate to systematically guide the selection among several architectural design alternatives:

- Its ability to specify and manage positive and negative interactions among goals [5, 16] allows the analyst to reason about design alternatives.
- Its capability to trace low-level details back to high-level concerns [6] is very appropriate to bridge the gap between architectural models and requirements.

These are the main reasons why the Goals Model has been introduced as an ATRIUM artefact to identify and describe the users’ needs and expectations, their relationships and how these can be met for the target system. With this aim, several building blocks are used (see section 3.1) that are related to each other by a set of special relationships (see section 3.2) via a refinement process that uses both kinds of elements to build the Goals Model.

3.1. Building Blocks for the Goals Model

As was stated above the GM provides a number of abstractions in terms of which constraints on the software system have to be defined. A key element employed in its construction is a goal. It is defined as an objective that the system-to-be should achieve [15], i.e., a constraint or obligation that the system should meet. In its definition it is stereotyped as Functional or Non-Functional, according to the type of need or expectation it refers to:

- Functional goals describe services that the system provides, i.e., the transformations the system performs on the inputs.
- Non-Functional goals refer to how the system will do these transformations, for instance, in terms of performance, adaptation, security, etc. We are highlighting them because they are especially meaningful in terms of software quality [21].

Additionally, other aspects have to be stated when a goal is defined. For instance, a set of preconditions and postconditions has to be identified. Preconditions establish which situations must hold before some operation is performed. Postconditions define the situations that have to be achieved after some operation. Their evaluations help us to determine the best design alternatives among those that satisfy the postconditions for the established goals.

Moreover, each goal has to be classified according to its priority, from very high to very low, for the system-to-be. This classification helps the analyst to focus on the important issues. These priorities can arise from several factors: organizational ones when they are critical to the success of the development, constraints on the development resources, etc.

![Figure 2 Visual notation for goals](Image 2 Visual notation for goals)

A textual notation for goals definition, along with a visual notation (Figure 2), is provided to deal with the previous aspects:

- **GOAL** GoalName
- **ID** identifier
- **TYPE** [Functional | NonFunctional]
- **DESCRIPTION** ShortDescription
- **PRIORITY** [Very High | High | Normal| Low, Very Low]
- **AUTHOR** authorName
- **PRE conditions**
- **POST conditions**

Aside from goals, other building blocks for the GM are the Operationalizations. When an analyst has refined the initial set of goals, he/she must offer a set of solutions that allow the system to achieve the established goals. These solutions provide architectural design choices for the target system which meet the users’ needs and expectations. They are called operationalizations because they describe the operation of the system, i.e., the system behaviour, to meet functional and non-functional requirements. Their textual notation, together with their visual notation (Figure 3), is:

- **OPERATIONALIZATION** OpName
- **ID** identifier
- **DESCRIPTION** ShortDescription
- **AUTHOR** authorName
- **PRIORITY** [Very High | High | Normal| Low, Very Low]

![Figure 3 Visual notation for Operationalizations](Image 3 Visual notation for Operationalizations)

It can be noticed that there is no section in the textual notation to define the alternative solutions to satisfy a given goal. In contrast, it is in the SM where these solutions are expressed. However, entities known as operationalizations are introduced in the GM to represent conceptually each solution so that relationships among the different alternatives can be established within the GM. Operationalizations imply a coupling between the SM and the GM, and traceability between operationalizations and a
specific view of the SM is achieved by means of identifiers.

Additionally, operationalizations have a special property called Priority to express how meaningful a given solution is for the system, i.e., its relevance for the system development. It is used to face the tradeoffs between operationalization choices in case there are conflicts among them.

We can observe in the textual notation that operationalizations are not stereotyped in a similar manner to goals. This is because the same solution can be associated to different goals, both functional and non-functional. Therefore, they are not specifically stereotyped.

### 3.2. Relationships: An Element in the Refinement Process

The stated building blocks, goals and operationalizations, are inter-related by means of a set of relationships. They are in charge of gluing the different elements to complete the model and enhance its cohesion. Moreover, their relevance is not only restricted to this gluing but also they allow the analyst to introduce the rationale of the system design. The decomposition of goals or how an operationalization positively or negatively contributes to a goal, can be defined via relationships.

They are applied via a stepwise refinement process which takes an informal set of requirements, stated in natural language, as an input to trigger the model definition, and using some of the proposals in [2] to identify goals. In such way, every refinement step provides a new set of goals and/or operationalizations to the model. Therefore, the analyst ought to deal with a reduced set of building blocks at each step.

There are two types of refinements that can be applied: intentional and operational. The former describes how a goal can be reduced into a set of sub-goals via AND/OR relationships. The latter depicts how a set of solutions can be provided for a goal, the analyst can introduce them in terms of the stated building blocks, goals and operationalizations.

An AND relationship (Figure 4 (a)) between a goal GoalX and a set of sub-goals G1, ..., Gk is established if the sub-goals have to be satisfied in order to satisfy GoalX. Its textual notation is described as:

\[
\text{AND}(\text{Goal1}, \ldots, \text{GoalN}) \Rightarrow \text{ACHIEVE GoalX}
\]

A goal GoalX is related to a to a set of sub-goals G1, ..., Gk via an OR relationship (Figure 4 (b)) if GoalX is satisfied if at least a sub-goal is satisfied. Its textual notation is described as:

\[
\text{OR}(\text{Goal1}, \ldots, \text{GoalN}) \Rightarrow \text{ACHIEVE GoalX}
\]

Every goal, which is too coarse-grained to be directly addressed by a solution, is refined in a set of subgoals which are a decomposition of the original one. Whenever a sub-goal is needed to achieve a goal, an AND relationship is established between them. On the contrary, if the sub-goal may optionally appear, then an OR relationship is established between them.

![Figure 4](image1)

**Figure 4 Visual notation for Goals Relationships**

Additionally, a CONFLICT relationship (Figure 4(c)) can be set up among two goals if an incompatibility appears between them, in other words, whenever the satisfaction of a goal prevents the satisfaction of another goal. It is described as:

\[
\text{CONFLICT Goal1, Goal2}
\]

An operational refinement deals with operationalizations and goals. The alternative solutions for each goal are established by means of this decomposition. There can be a large number of valid operationalization methods that are applicable to a goal. In such a case, it is up to the analyst to examine the impact of such methods on other types of requirements and decide on what and how many operationalizing methods must be applied via AND OPERATIONALIZE/ OR OPERATIONALIZE relationships. An AND OPERATIONALIZE relationship (Figure 5 (a)) relates the set of mandatory solutions for a goal. On the other hand, whenever several alternative solutions can be provided for a goal, the analyst can introduce them in terms of the OR OPERATIONALIZE relationship (Figure 5 (b)) relationship. Their textual notation is:

\[
\{[++]|[#]|-|--\} \text{Oper1, } \{[++]|[#]|-|--\} \text{Oper2} \Rightarrow \text{AND OPERATIONALIZE } \text{Goal1, Oper1} \} \text{OPERATIONALIZE } \text{Goal2, Oper1}
\]

![Figure 5](image2)

**Figure 5 Visual notation for Operationalize Rel.**

It is shown in this description that a set of symbols [++]|[#]|-|-- are used to stereotype the relationship. They denote how an operationalization collaborates to achieve a goal. Symbols ++ and + describe a positive collaboration, i.e., it provides a sufficient or partially sufficient solution, respectively, to satisfy the related goal.
Aspects, i.e., they are the Components (and connectors) are defined by a gluing of architectures, follows an aspect-oriented approach. We need to bear in mind that PRISMA, the selected model to instantiate concrete aspects in the end system. We need to bear in mind that PRISMA, the selected model to instantiate concrete aspects in the end system. This is determined within ATRIUM by means of the SM which scatters the aspects over the components and connectors by using the UCs and the MSCs. The UCs identify as actors the components, the subsystems or systems which are collaborating to operationalize a goal, i.e., a concern. As a result, the UCs where the involved actors are only components will be identified as aspects. In such a way, the set of UCs, where a component $C_1$ is involved, is the set of aspects that compose $C_1$.

3.3. Goals model and Aspects: How to Solve the Tangled Components

As was stated above, AOSD provides a mechanism to specify and relate concerns (solving the crosscutting). In this respect, these can be handled adequately avoiding that they appear tangled over the system and providing significant advantages in terms of understanding, maintenance and evolution. Concerns may or not result in concrete aspects in the end system. We need to bear in mind that PRISMA, the selected model to instantiate architectures, follows an aspect-oriented approach. Components (and connectors) are defined by a gluing of aspects, i.e., they are the "melting pot" for them. Therefore, it is necessary to identify what aspects are and which components (and connectors) are crosscut by them.

An issue arises when defining the term "concern", and more specifically, when deciding how it appears in our approach. It is difficult to answer this question because there is not a well-established understanding of the notion of concern. In AOSD, the term concern is understood as the so-called functional and non-functional properties. Our position is similar: the concerns of the system are like functional and non-functional goals. This answer may help to understand why the GM is so important in this work. The concerns are being established when we are identifying the goals (and sub-goals) of the system, by means of the iterative process stated above.

The concerns of the system are specified as nodes of the goals graph. Moreover, the GM assists us in identifying one of the main issues in AOSD: which concerns are being crosscut by others, i.e., which concerns are in conflict. The initial set of goals (concerns) could seem that interact in synergy, without any conflict. But the iterative refinement process can help us to identify if crosscutting appears. In this case, a conflict relationship is established between each pair of conflicting sub-goals. In order to solve these conflicts, the priorities defined over the relationships between goals and sub-goals determine the dominant sub-goal. A trouble may appear when several sub-goals show an equal priority. In such a case, the analyst must define a trade-off among them.

We have been speaking about the concerns of the system, but what about the aspects? Several works\cite{24} have stated that concerns may or not be aspects in the end system. This is determined within ATRIUM by means of the SM which scatters the aspects over the components and connectors by using the UCs and the MSCs. The UCs identify as actors the components, the subsystems or systems which are collaborating to operationalize a goal, i.e., a concern. As a result, the UCs where the involved actors are only components will be identified as aspects. In such a way, the set of UCs, where a component $C_1$ is involved, is the set of aspects that compose $C_1$.

4. Applying ATRIUM to a Case Study

Grid Applications \cite{11} support a new model of computing that has emerged from conventional distributed computing, but focusing on large-scale resource sharing and innovative scientific applications. This new type of computing environments raises new challenges with respect to coordination, namely the heterogeneity of the coordinated processes, the volume and volatility of the shared information and the potentially vast number of processes that require an homogeneous way of communication, abstracting the machine-dependent details. Moreover, several works \cite{4, 12} have recognized their need for dynamic self-adaptation to changing environments to deal with resource variability, system faults, etc.

Hoschek et al show in \cite{11} some of the exhibited requirements for a Data Grid. In particular, one of their real-world applications areas is the High Energy Physics (HEP) to allow a wide and distributed number of researchers to analyse generated data by large particle accelerators. HEP is used to design experiments, e.g. beams of protons accelerated in opposite direction, which are analysed in order to try to understand physical laws. An analysis that is carried out over a huge amount of data by interactive and batch processes but with a high performance in terms of time and space.

Moreover, a previous work \cite{12} showed how a set of Grid Computing Nodes exhibit some needs in terms of efficiency as CPU load, number of waiting processes, or available memory. Grid Computing Nodes gather this context information. An efficient task scheduling is needed in order to optimise the available resources, i.e., self-configurable or auto-adaptive applications which perform architectural evolution based on contextual information.
Several Functional and Non-Functional Goals can be established from these sentences. They provide us an oversimplified view of this kind of applications, but which will allow us to understand how GM is defined and its relationship with the concerns of the system-to-be. Specially, it will show how GM and SM are interleaved along the process.

**Iteration 1 - Step 1 Goals Model.** The initial set of Requirements is presented as five nodes in Figure 7 obtained from the previous sentences. *Data production* deals with experiments design. *End-User Analysis* with the required process to evaluate generated data. The *Performance* requirement is due to high efficiency required from this kind of applications. *Adaptation* appears because of the self-configurability or auto-adaptability that is needed. Finally, *Security* is introduced to deal with information protection.

**Iteration 2 - Step 1 Goals Model Definition.** Several subgoals can be identified in order to satisfy the previous ones, as it is shown in Figure 6. For instance the *Adaptation* goal can be intentionally refined up to *ContextAwareness* and *Interoperability*. The former gathers context information to adapt the system behaviour and the latter manages the heterogeneity of related elements. Because both sub-goals are required to achieve the Adaptation goal, an AND relationship has been established between them. In a similar manner, *Time* and *Space* are the intentional refinement of *Performance*. However, Space and Performance are OR-related because there is no high restriction in this sense.

**Iteration 3 - Step 1 Goals Model Definition.** Some of the identified goals can be operationally refined. ContextAwareness and ContextAcquisition are AND OPERATIONALIZE-related because the latter is a required solution for the former (Figure 8).

**Iteration 3 - Step 2 Scenarios Model Definition.** In order to operationalize the subgoal ContextAware, three actors can be identified: GridComputingNode, ClusterManager and LoadBalancer. The first one has to gather context information and distribute it to the LoadBalancer and the ClusterManager in order to optimise the resources. UC and the MSC, for this sub-goal are showed in Figure 9.

**Iteration 3 - Step 3 Collaborations Definition.** Because MSCs are closely related to collaboration diagrams it is very easy to get the latter. The result for the ContextAware goal can be observed in Figure 10. It is appreciated that the...
three components (GridComputingNode, LoadBalancer and Cluster Manager) have to be disguised with this goal (aspect). Moreover, to get a loosely coupling between the involved components, a connector is defined to aggregate and analyse the relationship between them.

**Iteration 3 - Step 4 Formalization.** The defined models will be formalized in this step, but a detailed description is outside of the scope of this paper.

**Iteration 3 - Step 5 Compilation.** According to the information provided by the previous models and the proto-architecture, and after validating via 3APL, the PRISMA model can be instantiated, describing the identified elements and the configuration that depicts their relationships (Figure 11). Both GridComputingNode, LoadBalancer and ClusterManager are shallow components that have to be fully described by the analyst. This proto-architecture can be influenced when the conflict relationship (Figure 8), established between ContextAwareness and Performance is resolved.

![Figure 11 A preview of the Instantiated PRISMA model](image)

We have used the previous example because the inherent dynamism of these applications. Nevertheless, the process can be instantiated in any domain where the software architecture has to deal with goals and allocation of responsibilities.

5. **Discussion and Related Work**

In this section, we briefly review the related works. Each of them provides the analyst with a different approach to the concurrent definition of requirements and architecture.

Brandozzi and Perry [3] have suggested the use of intermediate descriptions between requirements and architectures known as “architectural prescriptions”. These describe the mappings between requirements aspects and architectural descriptions. Nevertheless, it is not clear that the overhead of an additional description is practical. Recent work on software product lines and system families has also examined the relationship between architectures and requirements. The work has focused on identifying core requirements (identified through a process of requirements prioritising) and linking them to core architectures (identified by examining the stability of various architectural attributes over time).

Hall et al [9] show an extension to problem frames. Problem frames define requirements as dependencies among the system and the world outside of the computer. They help the developer to focus on the problem domain, establishing a clear separation between the world and the machine. This solution is amenable to perform iterative development due to the interaction among requirements, architecture, and design: each one informing about the others and vice versa.

Grunbaccher et al [8] explore the relationships between software requirements and architectures, and propose an intermediate model, CBSP (Component - System – Bus - Property), to depict the dependencies among the key architectural elements and the stated system requirements. These dependencies are established in a polling process where the involved architects classify each requirement with respect to its architectural relevance. Based on the CBSP model elements a proto-architecture can be derived with the selection of the appropriate architectural style. Therefore, it is a systematic approach, not an automated one, which is only applied in a specific context.

Lamsweerde [15] proposes a goal-oriented approach to architectural design based on the KAOS framework. It defines an iterative refinement process from functional specifications to an abstract architectural draft. It is refined to meet domain-specific architectural constraints. The main restriction shown by this approach is related to the use of non-functional requirements. They are only used to make decision among several alternatives but they are not as relevant as functional ones.

Liu et al [17] propose a similar approach by combining scenarios and goals-based models during architectural design. The GRL language is used to define the latter and the UCM notation is used for the former. The main problem that we observe is that both models are loosely coupled resulting eventually in a lack of consistence between them.

The main advantage shown by this approach with respect to the previous is related to the guided and semi-automated process support given from requirements to architectures. Also the coupling of requirements artefacts provides a better assessment of the design alternatives. Moreover, due to the traceability among requirements and architectures, along with the reflexive properties of PRISMA, running-systems can evolve over time while remaining compliant with their evolving requirements.

6. **Conclusions and Future Work**

Summarizing, in this paper we show how to address the iterative development of requirements and architectures during the development of software systems. A methodology, ATRIUM, that guides the analyst, from an initial set of requirements to an instantiated, architecture has been presented. It uses the strength provided by the coupling of scenarios and goals to systematically guide through the iterative process. Moreover, it allows the traceability among both artefacts to avoid lacks of consistency.

Additionally, requirements and architectures can evolve
iteratively and concurrently, in such a way that running-systems can dynamically adapt their composition and/or topology to meet their evolving requirements. It is granted by the reflexive properties of PRISMA and by the relationships between requirements and architectures that have been established along the process.

The GM, a requirement artefact used in this process, has been defined in this work. It has been designed to allow the analyst to reason about design alternatives by using the defined relationships. Additionally, the refinement process compels him/her to focus on a specific view of the system-to-be definition and, therefore, on a partial view of the architectural model, owing to its ability to trace low-level details back to high-level concerns.

There are questions and problems that remain open: the completeness of each scenario; the completeness of the set of scenarios; the compatibility and consistency of the architectural styles; the creation of the architectural structure or its transformation on the basis of NFRs. All these subjects constitute our future work.

7. References