

The FRIDA Model

Silvia de Castro Bertagnoli, Maria Lúcia Blanck Lisboa

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil – Voice: +55 (51) 3316-6803

{silviacb, llisboa}@inf.ufrgs.br

Abstract. *The goal of this paper is to establish a well-defined model, called FRIDA, which guides the developer through the process of software development. The main idea of this model is to determine a way to elicit and model both functional and non-functional requirements. Generally, the non-functional concepts are highly intermixed with the functional. In order to avoid that, our model is well founded on the aspect-oriented software development. The aspect is a complementary manner to describe the non-functional features related to any applications. In our approach, each NFR is represented with one or more aspects, and functional requirements are restricted to classes hierarchies. Moreover, the UML and some extensions of it were used to determine this model and allow the modeling of both objects and aspects involved in the system.*

Keywords: *Aspects, Non-Functional Requirements, Functional Requirements.*

1. Introduction

Today, the dominant paradigm to develop software systems is the object-oriented. The ideas involved in this paradigm are related to building software in which the problem is only decomposed in classes – dominant dimension ("tyranny of dominant decomposition") [17]. Hence, the class exhibits crosscutting structure, in which the concerns are scattered and tangled within classes [14]. Actually, the traditional OO paradigm has certain limitations, such as (i) the functionality is partially expressed by classes [10], and (ii) it is very difficult or impossible to encapsulate all concerns, essentially the non-functional ones [17]. Moreover, the non-functional concerns usually crosscut the functional units, i. e. transverse *Functional Requirements* – FRs [5].

Note that in OO the classes are commonly used to model and implement FRs, but to realize it with NFRs – *Non-Functional Requirements* – they lack a concept to express them. This situation happens due to a number of reasons: NFRs express quality attributes - not artifacts, they act upon other components or objects, or they can be considered as a "global" property of an application and thus can affect several requirements.

Several approaches in the literature [9] are pointed as the solution to deal with the NFRs. We choose the AOSD –

Aspect-Oriented Software Development, for the following reasons:

- (i) the goal is to provide mechanisms to identify, separate, represent and compose crosscutting concerns (for instance a NFR) [5, 13];
- (ii) the AOSD allows the crosscutting concerns to be better encapsulated as aspects [14, 19] (the NFRs are considered crosscutting concerns [13]);
- (iii) using several abstraction levels, AOSD, can be adopted at different stages of the software lifecycle [10];
- (iv) the AOSD is aimed as the solution for the problems involved in NFRs eliciting and modeling [1];
- (v) the aspect-oriented paradigm is indicated as a solution to the problems involved in NFRs [11];
- (vi) the aspect can be considered as a first-class element, and the software can be modeled in "2-dimensions" – the class and the aspect [7].

When a NFR is modeled since the initial phases of the process of development, the chances of success for the project are maximized [18]. On this scenario, we propose a model called FRIDA – *From Requirements to Design using Aspects* – in which the key objective is to guide the developer through the basic phases of the software lifecycle. The general lines of FRIDA concentrate on how the separated modeling of NFRs can aid in the FRs and the component analysis.

This paper is organized as follows. Section 2 concentrates on the FRIDA model. Section 3 presents the related works. Section 4 draws some conclusions and highlights some future works.

2. The FRIDA Model

In our approach, each and every NFR is not restricted to a single level. Our objective is to separate the NFRs from FRs, since it can improve both the analyses of the FRs and the generated components. FRIDA determines how the aspects will be identified and modeled during the software development process, i. e. how it is possible to model the NFRs. The general lines of this model are divided into phases, which are shown in the next subsections.

2.1 Requirements identification

The first step in this phase is a profound analysis of the problem. Based on this analysis it is possible to build the

use case diagram. For each use case occurrence, one template will be associated and will describe the use case in a detailed way.

Figure 1 presents the template used to describe each use case. It was constructed by combining several proposals [2, 8, 20]. We decide to use the template because it aids to describe the FRs in detail, and the FRs analysis is enriched.

Name	Descriptive phrase that names the use case.
Goal	Presents the goal of this use case.
Author	Person responsible for elaborating the use case.
Pre-condition	The state in which the system will be before the use case begins.
Post-condition	The state in which the system will be after the use case has been completed.
Primary actor	The key actor in the use case.
Secondary actor	Another actor(s) that realizes any action.
Priority	Used to indicate how this use case will be delivered to customer.
Primary Pathway	Description of the main event flow.
Alternate	Summarized description of the alternate event flow.
Exceptional	Summarized description of the exceptional flow of this use case.
Main	This section describes the main activities of the scenario of this use case. Observe that the focus here is on what must be done, and not how.
Variations	Here the steps that modify the main sequence steps are described.
Non-Functional	This section is reserved for appointing the generic NFRs related to the present use case: Performance: <resumed description> Security: <resumed description> Dependability: <resumed description>

Figure 1. FRs and NFRs Template

Note that in the last template section we introduce a short description to indicate the NFRs associated with the present use case. This description is summarized, i. e. vague and sometimes incomplete, and it needs more refinement in order to express an aspect.

The next step is to identify more precisely the NFRs required by each use case. In FRIDA, checklists are used to refine NFRs at early-stages of the development lifecycle.

2.2 NFRs Refinement

This step considers the generic NFRs, which are early available in the requirements elicitation phase (as shown in Figure 2). It operates on different levels of refinement, and tries to capture more precise information about the aspects involved. For this purpose, checklists are provided to ensure the completeness of these specific requirements.

Checklists provide a list of choices that can serve to guide the developer during the process by querying, advising and recalling previous issues about the NFR on focus [15]. For example, for each NFR being factored, the other NFRs linked to that particular use case are shown up.

	R ¹	N/A ²	P ³	Description
Dependability				
Reliability				
Is the system fault-tolerant?	<input type="radio"/>	<input type="radio"/>		
Does the system have exception handling or/and error recovery?	<input type="radio"/>	<input type="radio"/>		
Is any mechanism of redundancy necessary?	<input type="radio"/>	<input type="radio"/>		
Is quick recovery or startup capability necessary?	<input type="radio"/>	<input type="radio"/>		

Figure 2. Checklists Example

Considering that a use case can encompass more than one NFR, it is advisable to assign a priority to each one for further conflict resolution. Conflicts are not a concern at this phase, once a conflict will show up later.

We argue that the checklists approach is suitable to address the problems involved in NFRs. However, the support provided by this approach is not complete. Hence, the satisfactory way found to complement the modeling of NFRs is the use of NFRs lexicon, detailed in the next sub-section.

2.3 Lexicon Processing

After describing the FRs and refining the NFRs, it is possible that some NFRs are not correctly elicited. Hence, we decided to use a lexicon, as suggested by Leite [15], and a BNF – *Backus Naur Form* [16] – to define it. The lexicon is similar to a glossary. Our lexicon is concerned with keywords related to the problem domain in which the NFRs are used (Figure 3).

<NFR_Generic> ::= <performance> <security> <dependability>
<dependability> ::= <availability> <reliability> <integrity> <confidentiality>
<reliability> ::= <unit_reliable> <unit_rec> in <unit_cap> n <unit_t> <unit_rec> in <unit_cap> in <unit_t>
<unit_t> ::= ms milliseconds seconds hour day month year week
<unit_cap> ::= maximum minimum mean largest less than more than
<unit_reliable> ::= error failure fault
<unit_rec> ::= recovery restore retrieve backup

Figure 3. Quality Attributes BNF

¹ R = relevant to problem context, ² N/A = not ascribed, ³ P = priority of this present NFR

In FRIDA the lexicon is the starting point to process and analyze any use case description and other use cases related to the present one, such as “<<include>>” or “<<uses>>”, “<<extends>>” and generalization.

The detection of NFRs is realized over the present use case template and the related use cases. All words, derived words and expressions found in both of them are selected. The matching of any word defined in the lexicon with these words or expressions is selected as a candidate NFR. The developer is required to examine the candidate NFR and to decide if it is or is not a real NFR. Each confirmation is responsible for activating the correspondent checklists.

The checklists and the Lexicon are the selected techniques to accomplish the NFR analysis. We choose them to indicate and isolate the NFRs from the FRs, and because it turns the NFRs analysis become less abstract.

2.4 Conflicts Identification and Resolution

In FRIDA the identification and resolution of conflicts among NFRs is as important as the elicitation of NFRs in the early stages of the software development lifecycle [4].

The underlying idea of the identification of conflicts is to use a knowledge base in which NFRs conflicts are stored. As some NFRs can not be precisely measured, we have decided to choose an alternative way to represent the conflicts. The basic mechanism used in FRIDA to define a knowledge base is the use of rules on set theory [3] (Figure 4).

(1)	alteration: $V \rightarrow V$ alteration(x) = $x + \varepsilon$ where: V is a value set, $x \in V$ and $x \neq \perp$, $\varepsilon \in V$ and $x \neq \perp$, $\langle V, +, \perp \rangle$ is a group
(2)	r_i alter \Leftrightarrow alteration (x_{ri}) where: r_i is the requirement i , x_{ri} is the aggregated value, alteration is defined in (1)
(3)	r_i impact $r_j \Leftrightarrow ((r_i \text{ alter}) \Rightarrow (r_j \text{ alter}))$ where: $r_i \neq r_j$, alter is defined in (2)
(4)	r_i conflicts $r_j \Leftrightarrow r_i$ impact r_j where: $r_i \neq r_j$, impact is defined in (3)

Figure 4. Rules for NFRs

Examining these rules, it is possible to determine when two attributes are conflicting if the value of all rules is true. For instance, a possible strategy for security is confidentiality, which causes system overhead. This attribute is clearly conflicting with performance. Actually, if the security level increases, it can have an impact on performance. However, if there is no concern about security, it is possible to maximize the system performance. The customer can decide about these trade-offs.

In short, these rules can approve or refute conflicts among NFRs by using well-specified constraints. We represent these rules through a matrix structure, also called conflicts matrix [4], as illustrated in Figure 5.

	Latency	Throughput	Confidentiality	Integrity	Availability	Reliability
Latency					✓	✓
Throughput	✓				✓	✓
Confidentiality	✓	✓		✓		✓
Integrity	✓	✓	✓		✓	✓
Availability						✓
Reliability	✓	✓			✓	

Figure 5. Conflicts Matrix

Conflicting relationships are established in this matrix. The matrix analysis should be realized according to the following steps:

- (1) Each row is necessarily combined with each column;
- (2) Applying the rules for each previous combination;
- (3) If the response of all rules is true, one conflict is identified.

Note that this matrix presents only few conflicts, but the developer can expand them. However, the new conflict addition must consider the rules.

When a conflict is identified, we try to solve it by using priorities. A priority is assigned to each NFR. If the conflicting NFRs have the same priority, stakeholders are inquired about solutions for the conflict. Thus, the final decision on how the conflict will be solved is established through negotiation.

2.5 Aspect Extraction

This phase deals with the extraction and identification of crosscutting concerns, considering the complete use case diagram. These crosscutting concerns can be classified into two categories [3]:

- Global: the NFR is pertinent to the system as a whole; in other words, it is related to all use cases. For instance, the confidentiality requirements should be considered as global requirements, since they are used in all system functionalities;
- Partial: in this case the NFR is associated with some functional requirements, i. e., it belongs to some application parts.

A template results from this phase, which integrates the functional and non-functional aspects of the application. This template follows the ideas by Brito [5] and Araújo [2].

Basically, the essential features of this template are level (global/partial), priority, use cases list and requirements list. The use cases list serves to link the functional to the

non-functional requirements. It constitutes a convenient way for linking, because it will be indispensable for relating aspects and components later on. The requirements consist of enumerating the necessary requirements for the present aspect. Note that this feature is originated in the checklists description. Figure 5 depicts this template in detail.

Name	Descriptive word that names the aspect
Level	It is used to indicate the present aspect level
Description	The goal of the present aspect reported in this section
Priority	Priority is used to indicate how the aspect is important to the system under development
Use Cases list	This section of the template concentrates on enumerating the use cases related to this aspect
Requirements list	Here the requirements of the present aspect are indicated

Figure 6. Aspect Template

The next step for the development of the conceptual model is to determine the design diagram. Furthermore, it establishes a link between the requirements and the design.

2.6 Requirements Association with Design

Following the requirement specification and analysis, it is possible to start the subsequent phase – the design. In this phase, it is necessary to transform the concepts into classes, for these classes are progressively enriched with details along the project. This is an essential advantage of OO. The classes encountered in the analysis are preserved for the design stage. They certainly are enriched with behavior and new state, and still they should be complemented by auxiliary classes.

The developer may concentrate to find behavior and to refine the relationships between classes. Furthermore, in this phase new classes may be added or redefined, since the main objective here is to provide a hierarchy of classes concerned with software solutions, such as databases, graphical interfaces, and so on. In short, the developer determines the classes that are involved in the problem solution using well-known techniques. As the classes are defined, they are linked with the use case diagram elements.

Observe that several diagrams of UML are used in the design phase, but in our work only the class diagram will be considered. This step is responsible for discovering the classes, and their behavior and state.

The proposed requirements association is linked with the classes of the class diagram (UML) along with the use cases and/or actors. This specified link is similar to an html link.

Actually, it is responsible for establishing a reference between diagrams (and for their elements), and it binds the requirements to the aspects. Moreover, a link between diagrams is very important because it enables the

traceability of any requirements, either FRs or NFRs. Then it is possible to affirm that it is bi-directional, i. e., it is feasible to find the requirements origin of the diagrams and artifacts generated in the subsequent phases and vice-versa.

Each class identified by the developer shall be bound to at least one use case. If it is not possible to establish this bind, an inconsistency exists. One class only exists if it is related to any functional requirement. For each bound class, a syntactic description is constructed, following the template shown in Figure 7.

```

<concept_name = "value">
  <use_cases>
    <template_id> value </template_id>
    [<template_id> value </template_id> ...]
  </use_cases>
  <actors>
    <actor> name </actor>
    [<actor> name </actor> ...]
  </actors>
</concept_name>

```

Figure 7. Requirements and Design Link

In this section only the FRs are bound to the use case, and only they obtained a connection to the UML notation (graphical representation) using the class diagram. Note that the NFRs have not been visually represented yet. For that, we have developed the next sub-sections.

Note that in this step the focus is on the FRs analysis and a component design. In short, this step is independent from NFRs, i. e., the FRs analysis is realized, as the NFRs do not exist.

2.7 Visual Modeling of Aspects

Aspects as well as classes are divided into state and behavior, i. e., they can have fields, and methods respectively [21, 22]. The easiest and most adequate way for understanding is to represent them using UML extensions, also called stereotypes. Stereotypes are mechanisms used to visually describe aspects identified previously [21] (Figure 8).

```

<<aspect>>
  aspect_name
<<aspect_fields>> aspect_field
<<aspect_methods>> aspect_method

```

Figure 8. Aspect Stereotype

At this moment, it is also necessary to determine the joint-points of the aspects and to establish the pointcuts between one class and one aspect. The joint-point is defined using a textual description. The pointcut and advice have already been represented as the UML Operation meta-model element, as shown at the bottom of Figure 9.

The behavior of the pointcut is the joint-points combined by the operators "and", "or" or "not". A pointcut specifies one query type, in which the class hierarchy is the base for the queries. Pointcuts are statically written, but they are used to adapt the code in runtime. Another key

feature of this step is that each requirement of the present aspect (section 2.5, requirements list) will be validated by one specified method.

<pre> <<aspect>> aspect_name </pre>
<pre> <<aspect_fields>> aspect_field </pre>
<pre> <<aspect_methods>> aspect_method <<pointcut>> pointcut_name() <<advice>> after() <<advice>> before() <<advice>> around() </pre>

Figure 9. Pointcut and Advice Stereotype

After mapping the aspects and their elements, it is necessary to establish the relationship between classes and their respective aspects. The next sub-section shows the proposed way to realize it.

2.8 Combining Components and Aspects

At this moment of the design stage, the developer acquires the diagrams that represent the aspects and the classes. However, these elements are disconnected from each other. This step shows how the composition of the components and aspects can be automatically realized.

Using the link defined in sub-section 2.6, it is possible to determine which functional requirements are bound to each class. Analyzing the template defined in section 2.5, the NFRs linked with each aspect are found. Crosscutting this information it is feasible to identify which aspects are associated with each class. In this way, each class will be associated with one or more aspects to complete the system modeling.

3. Related Works

The growing interest in the AOSD has generated several works. Until very recently the focus of these researches concentrated on implementation and design levels. However, in the last years the interest in diffusing the AOSD to earlier activities involved in the software development lifecycle has increased.

Analyzing the literature it is possible to observe a great number of works that include the elicitation and modeling of NFRs. The traditional approach for modeling and organizing the software systems is based on the FR.

There are several proposals on how to deal with NFRs [1, 2, 4, 5, 6, 15, 19]. Nevertheless, there are few proposals that address the NFRs as aspects.

These proposals do not implement any automatic detection of NFRs, and the majority does not demonstrate how to transform the requirements into design elements. FRIDA offers automatic detection of NFRs and their further transformation into classes and their relationships. FRIDA also offers traceability, i. e. from a design element it is possible to discover the

requirement and actors that generated this element in any of the previous phases.

FRIDA also addresses the following concerns: (i) how to integrate the elicited NFRs with the models used in the subsequent phases, and (ii) how it is possible to model the NFRs using “2 dimensions” of decomposition. The objective of our work is to provide a systemic way to realize them.

4. Conclusions and Future Work

Since requirements direct the software development, they are crucial for quality [12]. As a consequence, both functional and non-functional requirements shall be identified as soon as possible and their elicitation must be accurate and complete. This work shows how to discover non-functional requirements that represent software quality attributes in order to identify the separate aspects, as required by the AOSD approach.

We propose to elicit the quality attributes that represent aspects using checklists and lexicons. Both checklists and lexicons help the developer to discover the global aspects - applied to the whole application- as well as the partial aspects to be applied to some parts of the application. The next step is the identification of the possible conflicts generated among the aspects and how to solve them. In FRIDA the identification and resolution of conflicts is as important as the elicitation of NFRs at the early stages of the software development lifecycle. We show the use of a conflicts matrix to automate this process whenever possible. The aspect extraction and the visual modeling of aspects are very important for our model, because they allow the creation of models that can be reused in the future. Moreover, the connection between diagrams that belong to distinct phases of software development allows a high level of traceability.

In short, a model as a whole can be classified into two visions: the functional (concentrates on the steps 1 and 6), and non-functional (described in the steps 2, 3, 4, 5 and 7). The objective of these steps is to provide a clear separation between FRs and NFRs.

The next steps in the model development, which are not within the scope of this paper, are:

- (i) Identify and solve aspect-aspect composition: regarding the composition between aspects. For example, to examine how collaborators aspects can be combined;
- (ii) Treat the crosscutting functional requirements: dealing with the FRs that crosscut other FRs, and deciding how they will be mapped – using a class or an aspect;
- (iii) Concept polymorphism: a concept in some cases is treated as functional requirements, but in others it can be considered as non-functional.

We are still working on these problems and trying to propose an adequate solution.

As a future work, we recommend a study on software architecture and the way it can be integrated into the

proposed model, aiming at the definition of a complete or almost complete model. Another work is related to the composition of aspects with aspects, i. e., to determine how it is possible to combine two aspects if they satisfy different FRs, or the same FR.

As the work progresses, a tool has been partially developed to accomplish the software development cycle from an *Aspect Oriented Programming* – AOP – perspective.

References

- [1] Araújo, J., et al. “Identifying aspectual use cases using a viewpoint-oriented requirements method”, In: *Aspect-Oriented Requirements Engineering and Architecture Design*, Boston, 2003.
- [2] Araújo, J., et. al. “Aspect-Oriented Requirements with UML”, In: *International Workshop on Aspect-Oriented Modeling with UML*, Germany, Sept., 2002.
- [3] Bertagnolli, S. C., Lisbôa, M. L. B. Aspect-oriented modeling of quality attributes. In: *International Conference on Computer Science, Software Engineering, Information Technology, e-business, and Applications, (CSITEA'03)*, 2003. (accepted paper).
- [4] Boehm, B., In, H. “Identifying Quality-Requirement Conflicts”, *IEEE Software*, March, 1996. p. 25-35.
- [5] Brito, I., et al. “A requirements model for quality attributes”, In: *Aspect-Oriented Requirements Engineering and Architecture Design*, Germany, Sept., 2002.
- [6] Chung, L., et al. “Non-Functional Requirements in Software Engineering”, Kluwer Publishing, 2000.
- [7] Clarke, S. “Composition of Object-Oriented Software Design Models”, Ph.D. Thesis, Jan. 2001.
- [8] Coleman, D. “A Use Case Template: Draft for discussion”, *Fusion Newsletter*, April 1998. On: http://www.hpl.hp.com/fusion/md_newsletters.html.
- [9] *Communications of ACM. Special Issue on Aspect-Oriented Programming*, 44 (10), 2001.
- [10] Elrad, T., Filman, R. E., Bader, A. “Aspect-Oriented Programming”, *Communications of the ACM*, 44 (10), Oct., 2001. p. 29-32.
- [11] Filman, R. E., Havelund, K. “The Effect of AOP on Software Engineering, with Particular Attention to OIF and Event Quantification”, In: *Workshop Software-Engineering Properties of Languages for Aspect Technologies*, 2003.
- [12] Gross, D., Yu, E. “From Non-Functional Requirements to Design through Patterns”, In: *International Workshop on Requirements Engineering: Foundation for Software Quality*, Jun. 5-6, Stockholm, Sweden, 2000.
- [13] Kiczales, G., et al. “Aspect-Oriented Programming”, In: *European Conference for Object-Oriented Programming, ECOOP, 1997, Proceedings...*, Lecture Notes in Computer Science 1241, Berlin: Springer-Verlag, 1997, p. 220-242.
- [14] Laddad, R. “I want my AOP !, Part 1”, In: <http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html>.
- [15] Leite, J.C.S.P., Oliveira, A.P.A. “A Client Oriented Requirements Baseline”, In: *IEEE International Symposium on Requirements Engineering*, IEEE Computer Society Press, 1995. p. 108-115.
- [16] Naur, P., Backus, J. W., et al. “Revised Report on the Algorithmic Language Algol 60”. *Programming Systems and Languages*, Edited by Saul Rosen, New York: McGraw-Hill, 1969.
- [17] Ossher, H., Tarr, P. “Using Multidimensional Separation of Concerns to (re)shape evolving software”. *Communications of the ACM*, v. 44, n. 10, Oct., 2001. pp. 43-50.
- [18] Pace, A., D., Campo M. “Analyzing the Role of Aspects in Software Design”, *Communications of the ACM*, 44(10). Oct., 2001. p. 66-74.
- [19] Rashid, A., Moreira, A., Araújo, J. “Modularization and Composition of Aspectual Requirements”, In: *International Conference on Aspect-Oriented Software Development*, Boston, 2003.
- [20] Reed, P. R. “Developing Applications with Java and UML”, Addison-Wesley. 2002, 463p.
- [21] Suzuki, J., Yamamoto, Y. “Extending UML with Aspects: Aspect Support in the Design Phase”, In: *Aspect-Oriented Programming Workshop*, (held in conjunction with the ECOOP 1999).
- [22] Zakaria, A., A., Hosny, H., Zeid, A. “A UML Extension for Modeling Aspect-Oriented Systems”, In: *International Workshop on Aspect-Oriented Modeling with UML*, (held in conjunction with the 1st Aspect Oriented Software Development Conference AOSD 2002), 2., Germany, Sept., 2002.