

AN EVALUATION OF THE FBDM FRAMEWORK DESIGN METHOD

N. Bouassida,

Assistant en Informatique de gestion
Nadia.Bouassida@isimsf.rnu.tn

H. Ben-Abdallah

Maître Assistant en Informatique
Hanene.benabdallah@fsegs.rnu.tn

A. Ben Hamadou

Professeur en Informatique
Abdemajid.Benhamadou@isimsf.rnu.tn

Adresse professionnelle

Laboratoire LARIM, ISIMS, Route Mharza km 1.5, 3018, Sfax, Tunisie

Résumé : Cet article présente des travaux pertinents à la réutilisation architecturale. D'une part, il présente la méthode de conception de frameworks FBDM (Framework Based Design Method). D'autre part, il évalue FBDM à travers une étude de cas dans le domaine des éditeurs graphiques. La méthode FBDM offre un langage de conception, un processus de conception et un outil CASE. Le langage de FBDM est un profile UML, qui étend UML avec des annotations graphiques afin de distinguer entre le noyau du framework et les points de variation. Le processus de conception de FBDM génère un framework d'une manière semi-automatique en unifiant un ensemble d'applications dans le domaine du framework; le développeur du framework doit décider de la complétude de quelques relations faisant partie des points de variation. La méthode FBDM est évaluée à travers un framework d'éditeurs graphiques. Elle compare le framework généré par FBDM avec le framework populaire JHotDraw. De plus, l'évaluation utilise des métriques de conception pour examiner la qualité du framework généré et l'apport de la notation de FBDM.

Summary : This paper presents work pertinent to architecture reuse. On one hand, it presents the framework design method FBDM (Framework Based Design Method) which offers a design language, a design process and a CASE toolset. On the other hand, it reports on an experimental evaluation of FBDM in the domain of graphical drawing editors. The design language of FBDM is a UML profile that extends UML with graphical annotations to distinguish between a framework core and hot-spots. The FBDM design process generates a framework semi-automatically by unifying a set of applications in the framework domain; the developer is probed to decide on the completeness of some relations. The second part of this paper uses a graphical drawing editor framework to evaluate FBDM. It compares the framework generated by FBDM to the popular JHotDraw framework. In addition, it uses design metrics to examine the quality of the generated framework and the usefulness of the notation of FBDM.

Mots clés : Réutilisation, conception de frameworks, métriques de conception, FBDM.

Keywords : Reuse, framework design, design metrics, FBDM.

An Evaluation of the FBDM Framework Design Method

1 – INTRODUCTION

As computerized systems became larger and increasingly complex, the benefits of reuse became irrefutable. In fact, the software engineering community has been applying several object-oriented based reuse techniques ranging from class and component libraries (c.f., (Meyer, 1988) (Szyperski, 1996)) to, more recently, generic architectures (a.k.a. frameworks (Johnson, 1998)) and evolving models (c.f., MDA (OMG, 2001)).

The work presented in this paper is an academic contribution to architectural reuse through frameworks. An object-oriented framework offers the skeleton of applications in a given domain, that can be customized by an application developer (Fayad, 1998). Hence, it allows the reuse of design models, code and valuable domain expertise (Johnson, 1998). Therefore, it ensures an increased productivity, a shorter development time and a higher quality of applications.

Overall, a framework is composed of immutable parts, called *frozen-spots* or *core*, and adaptable parts, called *hot-spots*. A frozen-spot describes the typical software components and is, therefore, present in any application generated from the framework. On the other hand, a hot-spot allows the framework extension to a particular application, and helps in tracing the evolution of a given design.

One of the problems impeding the widespread use of frameworks is the difficulty of their development, combined with the lack of a standard notation that helps to identify the hot-spots. This problem motivated several design notations based on UML (c.f., (Pree, 1994) (Fontoura et al., 2000)) and a few design processes (c.f., (Schmid, 1997)). However, none of the proposed notations distinguishes between the core of the framework and the hot-spots and none of them relies on a formal semantics. Moreover, none of the design processes proposed in the literature gives rules that guide the generation of the framework and that determine its different components.

Clearly identifying (both finding and denoting) the core and the hot-spots of a framework is essential in

understanding and reusing the framework. This motivated our work in proposing a framework design method, called FBDM (Framework Based Design Method, that offers a design language, called F-UML with a precise semantics (Bouassida et al., 2003), a design process (Bouassida et al., 2002) and a CASE tool, called F-UMLTool (Bouassida et al., 2004). The F-UML language is a UML profile that increases the expressiveness of UML with framework specific concepts and that guides a framework design reuse. It adds tags and graphical annotations to UML diagrams; the extensions help to distinguish visually between the core of a framework and its hot-spots and guide the user in deriving a specific application in the framework domain. On the other hand, the FBDM design process is a bottom-up process that generates a framework design by applying a set of unification rules to concrete application designs. In addition, it generates a framework with a minimum intervention from the developer; this latter is probed only to decide on the completeness of some structural relations (inheritance, composition,...), a decision often requiring domain-specific knowledge.

After an introduction to the FBDM method, this paper presents an experimental evaluation of FBDM in the domain of graphical drawing editors; the choice of the domain was motivated by the presence of a popular, mature and open source framework in this domain (JHotDraw (Gamma, 2000)).

The FBDM evaluation was conducted in two phases. First, an intrinsic evaluation tested the completeness of the design unification rules and the quality of the generated framework. Secondly, a comparative evaluation tested the correctness of the design process and evaluated the usefulness of the F-UML notation.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 presents the framework design method FBDM. Section 4 presents the three graphical drawing applications as well as the framework generated through FBDM. Section 5 presents the evaluation metrics used and reports on the results of our two-phase evaluation.

Section 6 concludes with a summary of the presented work and outlines future works.

2. RELATED WORK

Classical design languages (e.g., UML) and processes (e.g., OMT (Rumbaugh, 1991)) lack concepts to determine and express the variability in frameworks. This motivated several researchers to propose design languages and processes specifically for frameworks. We next review framework design languages and processes that are based on the *de facto* standard language UML.

2.1 – UML-based Framework Design Languages

Fontoura et al. (Fontoura et al., 2000) propose a UML profile for frameworks, called UML-F, where a design is expressed by a class diagram and a set of sequence diagrams. This notation extends the two UML diagrams by *presentation tags* (e.g., complete, incomplete), *basic modeling tags* (e.g., fixed, application, framework) and *essential pattern tags* (e.g., FacM-Creator, FacM-ConcreteCreator). The added tags are used to mark, essentially, the complete/incomplete parts and the variable parts in the diagrams and the roles of diagram elements. In this notation, the extended class diagram represents the framework classes and relations. However, according to the tag definitions, this notation represents only the whitebox hot-spots; the second type of hot-spots, called blackbox (Schmid et al., 1997), are therefore not expressible in this notation. In addition, several tags are complementary and thus are redundant (e.g., complete and incomplete, application and framework). Furthermore, the pattern tags combined with the presentation tags could overcharge the class diagram and impede the design understanding.

Sanada et al. (Sanada et al., 2002) present an UML extension that “aims to be comprehensive and well defined”. Most of their proposed extensions have already been defined by Fontoura (Fontoura, 2000); the essential difference is the constraint *covariant* which shows that adding a subclass to a certain class might result in adding a subclass to another class in the class diagram.

Riehle (Riehle, 1998) proposes a role modeling language that adapts the OORAM methodology (Reenskaug, 1996). The proposed language represents a framework through a class model with an *extension-point class set* (places of extension), a *built-on class set* (the framework interface) and a *free role type set* (usages of the framework by other frameworks). This language focuses more on framework composition than framework adaptation. For instance, it does not visually distinguish between extension-point classes and frozen classes

in the framework. Therefore, one cannot easily recognize the whitebox and blackbox hot-spots.

Overall, the proposed UML-based design languages for frameworks lack the visual distinction between the core and the two hot-spot types. This distinction is essential in understanding and reusing a framework. In addition, none of the proposed languages relies on a formal semantics. This latter is vital in analyzing a framework and validating its reuses.

2.2 – Framework Design Processes

A design process for frameworks should systematically help to identify and derive a framework core, blackbox and whitebox hot-spots. It could follow either a top-down or bottom-up strategy. Bottom-up design works well where a framework domain is already well understood, for example, after some initial evolutionary cycles. In this case, the design process starts from a set of existing applications and generalizes them to derive a framework design (c.f., (Koskimies, 1995), (Fontoura, 2000)). On the other hand, top-down design is preferred when the domain has not yet been sufficiently explored. In this case, the design process starts from a domain analysis and then constructs the framework design (c.f., (Aksit, 1999)).

Koskimies and Mossenback (Koskimies et al., 1995) propose a two-phase bottom-up framework design process. The first phase, called *problem generalization*, incrementally generalizes a representative application in the framework domain into “the most general” form. In the second phase, called *framework design*, the generalization levels of the previous phase are considered in a reverse order, leading to an implementation for each level. The last step in the second phase applies the resulting framework to the initial application. This design process does not provide for reuse guidelines; that is, it does not clearly identify nor does it guide the designer in finding the framework core and hot-spots.

Schmid (Schmid et al., 1997) decomposes the framework design process into three steps: 1) design of a class model for an (arbitrary) application in the framework domain; 2) analysis and specification of the domain variability and flexibility, i.e., identification of the hot-spots; and 3) generalization of the class model by applying a sequence of transformations that incorporate the domain variability. This design process leaves the identification of hot-spots, during the second step, to the developer’s expertise.

Fontoura et al. (Fontoura et al., 2000) propose a design process that considers a set of applications

as viewpoints (i.e., perspectives) of the domain. The process informally defines a set of unification rules that describe how the viewpoints can be combined to derive a framework. This design process neither distinguishes between the two hot-spot types, nor does it specify the object interactions. In addition, this process does not address semantic issues in the unified applications (e.g., synonyms, homonyms,...); it supposes that all the semantic inconsistencies between the viewpoints have been solved beforehand.

3. THE FRAMEWORK DESIGN METHOD FBDM

A framework design consists in a framework design notation and a framework design process that guides the development of a framework. On one hand, the framework design notation must provide for a means to describe the framework static structure (the classes and their relations), the core, and the whitebox and blackbox hot-spots. In addition, it has to show explicitly the collaborations between objects instantiated from the framework classes, and to clarify object responsibilities. On the other hand, the framework design process has to provide for design rules helping in the design of frameworks and to provide for a methodical, systematic way to design frameworks. In addition, it must guide the user in determining the core and hot-spots of the framework.

These reasons motivated us to propose the FBDM design method with its design language F-UML and process.

3.1 - The Framework Design Language F-UML

A framework design expressed in the F-UML language is composed of four UML extended diagrams:

- A use case diagram that specifies the framework scope, objectives and domain limits. Table 1 summarizes and explains the F-UML extensions added to UML.
- A class diagram that describes the static architecture of a framework. Table 2 presents the F-UML notation for the class diagram.
- A pattern diagram that shows the design patterns (Gamma et al., 1995) and meta-patterns (Pree, 1994) in order to delimit the roles of the framework classes.
- A set of sequence diagrams that describe object interactions. The extensions to the

UML sequence diagram are described in Table 3.

In F-UML, a blackbox hot-spot represents a component that can be selected (without modification) in an application derived from the framework. On the other hand, a whitebox hot-spot can be selected and modified to meet the special requirements of an application derived from the framework. For example, a whitebox hot-spot class can be modified by specializing it (i.e., adding an inheriting class) adding/removing an attribute/operation, or redefining one of its operations. Note that, since the use cases represent the design at a high level of abstraction, they can not include whitebox hot-spots.

Being a UML profile, the F-UML language could be adopted easily by the UML community. In addition, having a precise, formal semantics (Bouassida et al., 2003) F-UML can be combined with a formal method that allows the verification and validation of framework designs. Currently, the F-UML semantics is expressed in Object-Z (Smith, 2000) and verification is conducted with the Z/Eves theorem prover (Bouassida et al., 2005).







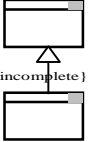
Notation	Explanation
	a framework core
	a framework blackbox hot-spot
	a framework whitebox hot-spot
	a re-definable (virtual) method in a whitebox hot-spot
	a method with an undefined signature
	an adaptable class interface (whitebox hot-spot)
	the framework may be adapted by adding other related classes (inheritance, composition,...)

Table 1. F-UML class diagram notation


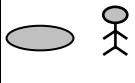
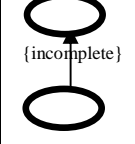
Notation	Explanation
	a framework core use case (actor)
	a framework hot-spot use case (actor)
	to show that it is possible to add an inheriting actor (use case) in an application adapting the framework

Table 2. F-UML use case diagram notation

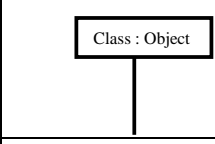
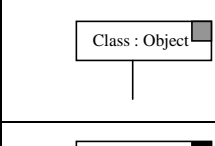
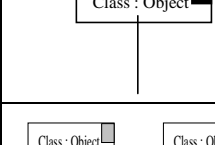
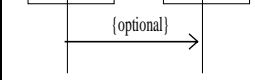
Notation	Explanation
	a framework core object
	a framework whitebox hot-spot
	a framework blackbox hot-spot.
	the message may not exist in an application reusing the framework

Table 3. F-UML sequence diagram notation

3.2 - The FBDM design process

The FBDM design process is a bottom-up process that takes several applications in a given domain and applies a set of unification rules to derive a framework in the F-UML notation. (Roberts (Roberts 1996) states that three applications sufficiently represent their domain). In addition, the FBDM design process helps the framework designer to structure the framework by determining automatically its core and hot-spots.

Overall, the design process is based on architectural and noun comparisons. Architectural comparisons

examine the structure of the applications defined through the different relations (composition, aggregation ...). Since a framework offers a basic architecture for the derived applications, architectural comparison of the unified applications is, therefore, a reasonable approach.

The second type of comparisons used in the FBDM design process is noun comparisons. These latter examine the nouns attributed to the different entities (actors, use cases, classes, attributes, methods ...) and, thus, deal with the semantics included in the applications through naming. These comparisons rely on the hypothesis that nouns are well chosen to reflect the roles of software entities. More specifically, they use semantic relationships (e.g., equivalence, generalization-specialization, composition) between nouns of classes, attributes and methods that appear in the applications to be unified. In addition, the unification rules compare the method signatures, which, combined with the noun comparison, give an insight on the dynamic behavior (or roles) of the classes.

In the remainder of this section, we note a class C in an application A_i as C_{A_i} , a use case U in an application A_i as U_{A_i} , and an actor A in an application A_i as A_{A_i} .

3.2.1 Unification of the use case diagrams

To design the framework use case diagram, the unification process first extracts use cases (actors) *common* to all of the applications and puts them as the framework core. Secondly, it extracts the different use cases (actors) and puts them as blackbox hot-spots. For this, it uses the following four semantic relations:

- $N_{equiv}(A_{A_1}, \dots, A_{A_n})$ means that the actor names are either identical or synonym.
- $N_{var}(A_{A_1}, \dots, A_{A_n})$ means that the actor names are a variation of a concept, e.g., employee-contractual, employee-permanent, employee-vacationer.
- $Gen_Spec(A_{A_1}; A_{A_2}, \dots, A_{A_n})$ means that the name of the actor A_{A_1} is a generalization of the names of A_{A_2}, \dots, A_{A_n} , e.g., $Person_{A_1}$ - $Employee_{A_2}$.
- $N_{dist}(A_{A_1}, \dots, A_{A_n})$ means that the name of the actor A_{A_1} has neither an equivalent nor a variation, nor a generalization in the other applications.

The design of the framework use case diagram through unification is guided by the six rules depicted in Figure 1.

3.2.2 - Unification of the class diagrams

To design the framework class diagram, the unification process determines the semantic correspondence between the classes, using the following criteria:

- class name comparison criteria to compare semantically the names;
- attribute comparison criteria to compare the names and types of the attributes; and
- operation comparison criteria to compare the names and signatures of the operations.

Note that, when comparing the set of attributes and methods of two classes, the unification rules take into account the interpretation of the inheritance relation. Thus, they consider the set of attributes (operations) of a class as those contained in the given class augmented with the set of attributes (operations) of the super classes from which it inherits.

Class name comparison criteria: They express the linguistic relationship among class names that belong to different applications. We defined five types of relations between classes:

- $N_equiv(C_{A1}, \dots, C_{An})$, $N_var(C_{A1}, \dots, C_{An})$, $Gen_Spec(C_{A1}; C_{A2}, \dots, C_{An})$, and $N_dist(C_{A1}, \dots, C_{An})$ are defined in a similar manner to their counter parts for the use case diagram; and
- $N_comp(C_{A1}; C_{A2}, \dots, C_{An})$ which means that the class name C_{A1} is linguistically a composite of the components C_{A2}, \dots, C_{An} , e.g., $House_{A1}$ - $Room_{A2}$.

Attribute comparison criteria: They express the relationship between the attribute names and the attribute types of application classes. They are grouped into four categories:

- $Att_equiv(C_{A1}, \dots, C_{An})$ implies that the classes have identical or synonym attribute names with the same types.
- $Att_int(C_{A1}, \dots, C_{An})$ implies that C_{A1}, \dots, C_{An} have attributes in intersection.
- $Att_dist(C_{A1}, \dots, C_{An})$ implies that no attribute of C_{A1}, \dots, C_{An} is common with the attributes of the others.
- $Att_conf(C_{A1}, \dots, C_{An})$ implies that there exists at least one attribute of C_{A1} with a name equivalent to the attributes of C_{A2}, \dots, C_{An} but their types are different.

Operation comparison criteria: Operation comparison consists in comparing the operation names and signatures (returned types and parameter types). The operation comparison criteria $Op_equiv(C_{A1}, \dots, C_{An})$, $Op_int(C_{A1}, \dots, C_{An})$, $Op_dist(C_{A1}, \dots, C_{An})$, and $Op_conf(C_{A1}, \dots, C_{An})$ are defined in a similar manner to the attribute comparison criteria.

The class diagram unification process is depicted in Figure 2. (For a detailed description of the unification rules, the reader is referred to (Bouassida et al., 2002). As illustrated in Figure 2, the unification rules determine automatically the framework core and hot-spots. The designer is probed only to decide on the completeness of some relations (Rule 3.d and 7). This information is, in fact, application dependant and requires domain knowledge.

In addition, in Figure 2, **Rule 5** deals with the generalization-specialization relation in a manner similar to N_Comp relation of **Rule 4**. Furthermore, **Rule 3.b** may add new inheriting classes to the framework. The addition depends on the significance of the number of attributes and methods in an inheriting class C with respect to another class C' . This is defined using the ratio R_{sig} :

$$R_{sig}(C, C') = \frac{\text{number of attributes and methods that belong to } C \text{ and } C'}{\text{number of attributes of } C + \text{number of methods of } C}$$

Informally, a class C has a significant number of attributes and methods with respect to a class C' , if R_{sig} is greater than a fixed threshold (e.g., 50%) that can be fixed by the framework designer. This ratio fixes the level of details the framework designer would like to include in the framework.

Further, **Rule 6** adds or ignores hot-spot classes according to the domain coverage ratio R_{dc} :

$$R_{dc}(C) = \frac{\text{number of occurrences of } C \text{ (or its variations or its equivalents) in } A1, \dots, An}{\text{number of applications}}$$

Informally, this ratio is used to determine the reuse potential of a class. If a class is present in several applications, then it covers an important space of the framework domain; thus, it must be present in the framework hot-spot. On the other hand, if a class is present in few applications, it is too application specific; thus, if it is added to the framework, it may complicate unnecessarily the framework comprehension.

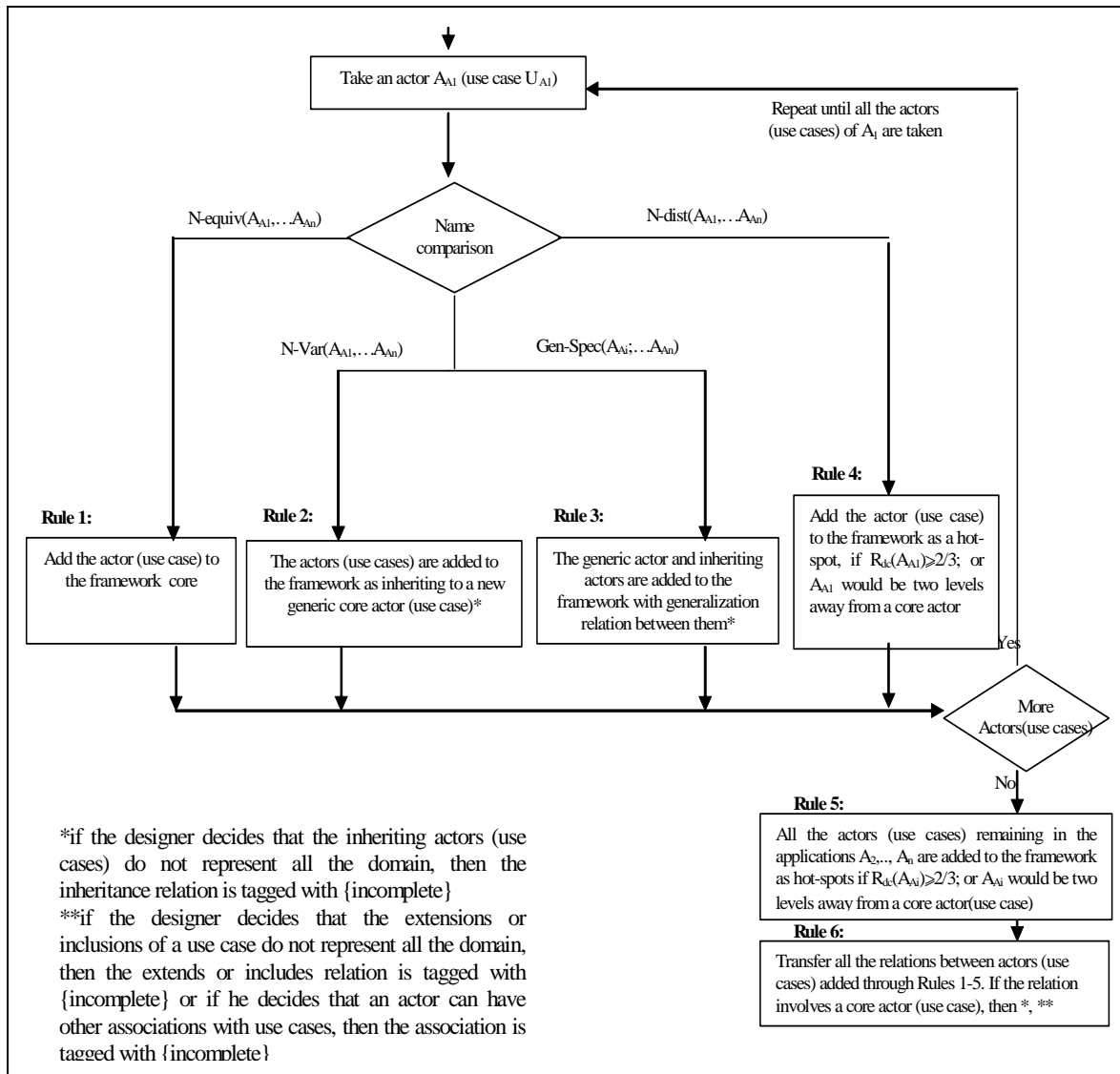


Figure 1: Design of the use case diagram

3.2.3 - Unification of the sequence diagrams

The framework sequence diagrams are obtained by the unification of the sequence diagrams which represents semantically equivalent scenarios. Briefly, the unification process takes the “union” of all the sequence diagrams of the given applications and marks any message as *{optional}* if it does not appear in all the applications. Any sequence diagram that does not have equivalent diagrams in the other applications being unified is transferred to the framework with all its messages marked *{optional}*.

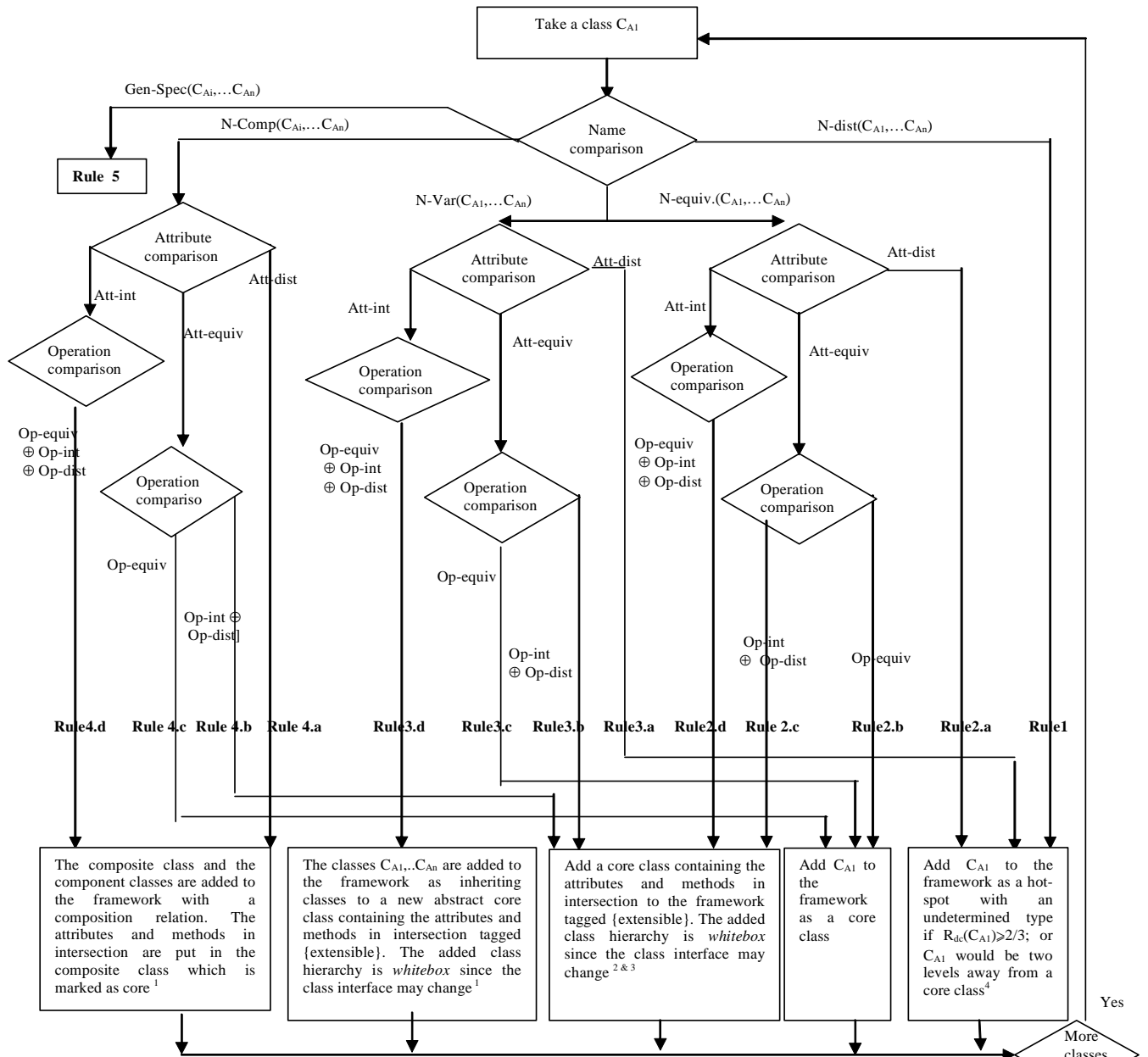
For more details on the six unification rules of the sequence diagrams, the reader is referred to (Bouassida et al., 2002).

Finally, we note that the FBDM design process can be optimized in two ways. First, the unification can start with the application having the minimum

number of elements (i.e., use cases/actors, classes, and messages). Secondly, in the unification rules of the class diagrams, the comparisons can be limited to “significant” attributes and operations, i.e., those that are not omni present in the classes; for example, the creator and destructor operations in a class.

4 - THE GRAPHICAL DRAWING EDITOR CASE STUDY

Two motivations were behind our choice of the graphical drawing editor domain. First, an open-source and mature framework already exists (JhotDraw (Gamma et al., 2000)) and, secondly, several derived applications also exist. The availability of the framework and its applications allowed us to conduct both an intrinsic and a comparative evaluation of FBDM.



¹ The designer must decide on the completeness of the relation. If he decides that the component (inheriting) classes do not represent the entire domain, then the composition (inheritance) relation is tagged with *{incomplete}*.

² New classes inheriting from the added class could be added according to the following rule: for each application, if its class has a "significant" number of attributes and methods (with respect to the already added class), then an inheriting class is added to the framework with the additional attributes and methods.

³ If $Op\text{-Conf}(C_{A1}, \dots, C_{An})$, thus the method in conflict has a corresponding method in the framework core class with the same name, and an undefined signature, it is a virtual method. If $Att\text{-Conf}(C_{A1}, \dots, C_{An})$, thus the attribute in conflict has a corresponding attribute in the class of the framework core that has the same name and the more general attribute type.

⁴ The domain coverage ratio $R_{dc}(C) = \text{number of occurrences of } C \text{ (or its variations or its equivalents) in } A_1, \dots, A_n / n$

Rule 6: Each class C remaining in A_2, \dots, A_n is added to the framework as an undetermined hot-spot if the domain coverage ratio $R_{dc}(C) > 2/3$; or C would be two levels away from a core class⁴

Rule 7: Transfer all the relations between classes to the framework. If the relation involves a core class, then¹

Rule 8: Visit all hot-spot classes C with an undetermined type. If C contains virtual or undefined methods or if one of its inheriting classes is whitebox, then mark C as a whitebox, otherwise mark C as a blackbox. If C has a relation with a core class, then¹

Figure 2: Class diagram design

Recall that the FBDM design process starts from existing applications. For this, we have chosen the following three applications (due to the availability of their source code): JARP, a graphical composer for petri nets (JARP, 2001); JOONE, a Java framework to create, train and run neural networks (JOONE, 2001); and RENEW, an editor for drawing reference nets (RENEW, 2001).

Before starting our evaluation, we have re-engineered the Java source code of the three graphical drawing editor applications with a help from the Rational Rose tool (Rational, 2004). In addition, we manually specified their use case diagrams and sequence diagrams by examining their documentation. Due to space limitations, we next give a quantitative description of the three applications and focus on describing the framework generated through FBDM.

4.1 – The Unified Applications

JOONE has a use case diagram composed of one actor and 14 use cases related by 2 “*extends*” relations, 5 “*includes*” relations and 6 associations. Its class diagram is composed of 242 classes interrelated by 132 inheritance relation, 35 associations and 48 aggregation. It has 15 sequence diagrams.

On the other hand, the JARP application has a use case diagram composed of one actor and 13 use cases related by 5 “*extends*” relations, 3 “*includes*” relations and 5 associations. Its class diagram is composed of 175 classes related by 123 inheritance relations, 33 associations and 24 aggregations. It has 12 sequence diagrams.

As for the application RENEW, it has a use case diagram composed of one actor and 21 use cases related by 3 “*extends*” relations, 11 “*includes*” relations and 5 associations. Its class diagram is composed of 328 classes related by 243 inheritance relations, 191 associations and 17 aggregations.

4.2 – The Generated Framework

Figure 3 shows the use case diagram of the framework generated by unifying the three applications in F-UMLTool. Similar to the unified applications, it has one core actor. In addition, it has 27 use cases marked as blackbox hot-spots

(e.g., “create a neural network Figure” and “create a petri Net Figure”); these use cases were not present in all of the unified applications and, thus, represent particular application-specific functionalities. Their presence in the use case diagram gives the designer an idea on possible adaptations of the framework and guides him/her in the reuse. Furthermore, the generated use case diagram contains seven use cases marked as core (e.g., “create figure”); these latter represent the basic functionalities of any graphical drawing editor application, which also the case of the three unified applications.

Figure 4 partially shows the generated class diagram of the framework. The complete class diagram contains 220 classes, among which 38 are core, 4 are both core and whitebox, 76 are blackbox and 102 are whitebox classes.

Examining closely the generated class diagram, we found that among the classes present in the three applications, we have the classes Figure, AbstractFigure, AttributeFigure and CompositeFigure; thus, these classes are part of the framework core. In addition, among these classes, the classes AbstractFigure, AttributeFigure and CompositeFigure contain virtual methods; that is, they could be refined either via inheritance or by overriding some of their methods. For this, they are marked as both whitebox and core classes.

During the generation of the framework class diagram, F-UML Tool decided automatically on the type of the hot-spot classes (e.g., AttributeFigure, PolylineFigure are whitebox, PetriPlace and PetriTransition are blackbox). It prompted us to decide on the completeness of 16 inheritance and composition relations. Among these relations, we cite the inheritance relation between AttributeFigure and RectangleFigure, EllipseFigure, TextFigure; since the inheriting classes do not represent the entire domain, we have decided to tag this relation {*incomplete*}.

Figure 5 presents an example of the generated sequence diagrams, for the “Create a figure” use case. Some objects (e.g., PetriPlaceImpl and LayerFigure) are blackbox in conformance with the class diagram. The message LayerFigure is tagged optional since it appears only in one application among the three unified ones.

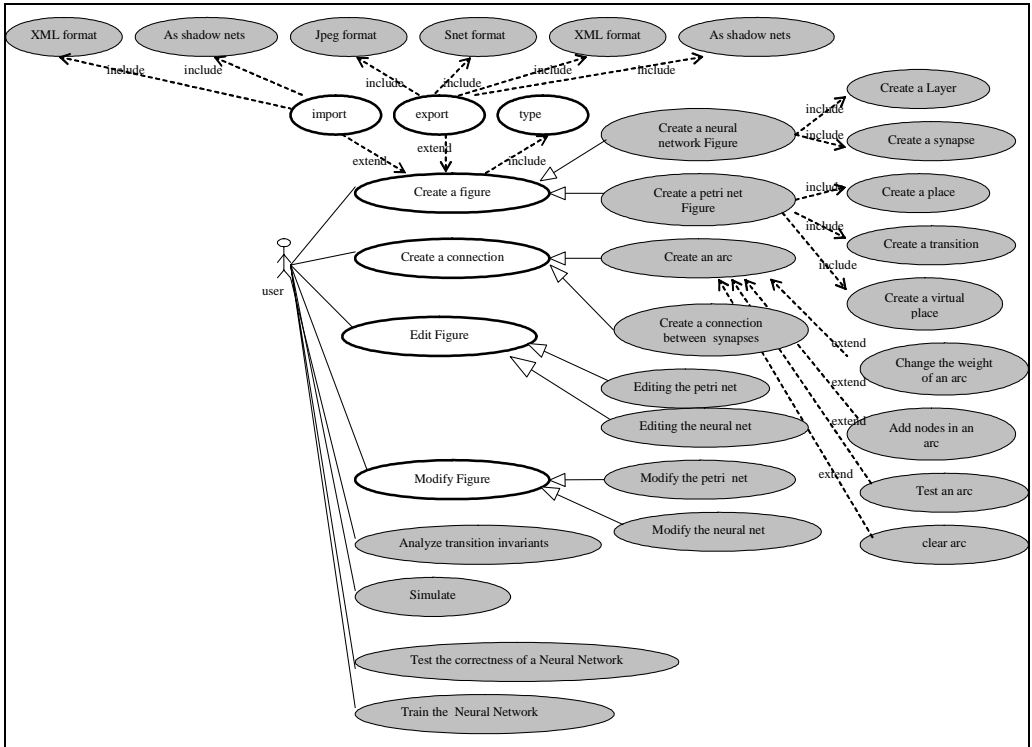


Figure 3: Use case diagram of the generated graphical drawing editor framework

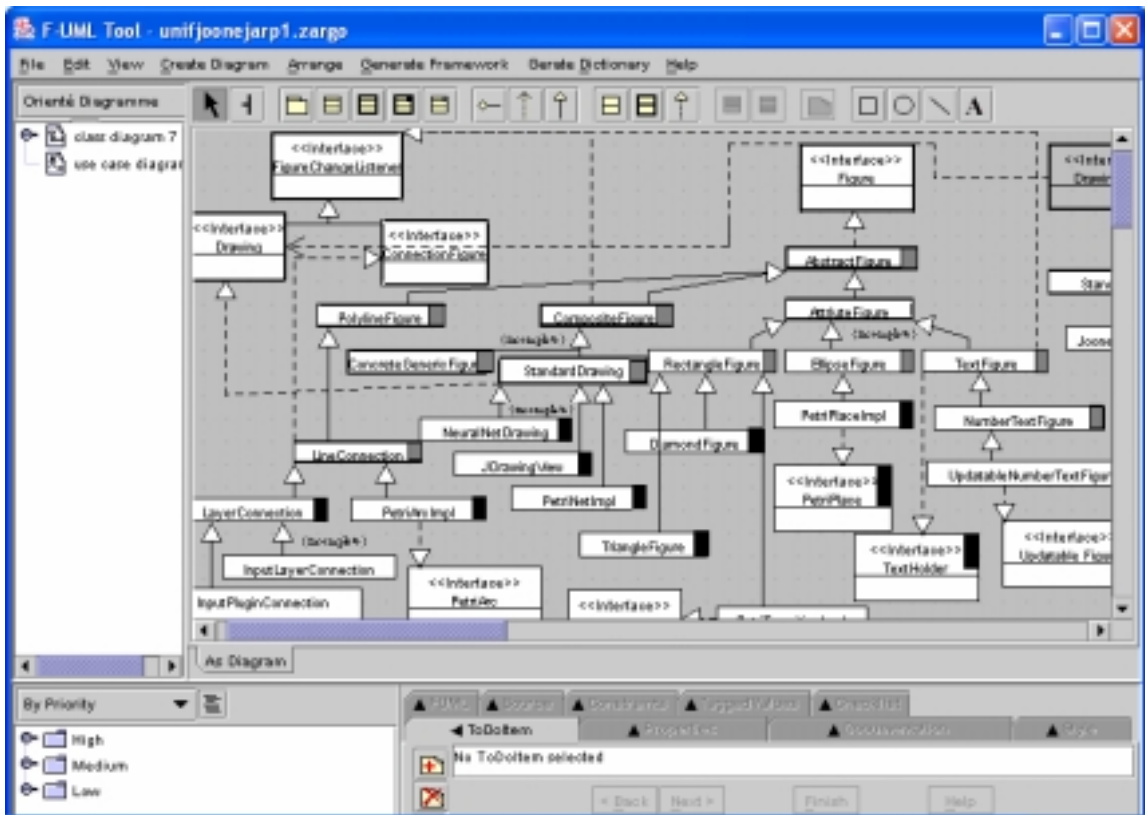


Figure 4: Class Diagram of the generated Graphical Drawing Editor Framework (partial view)

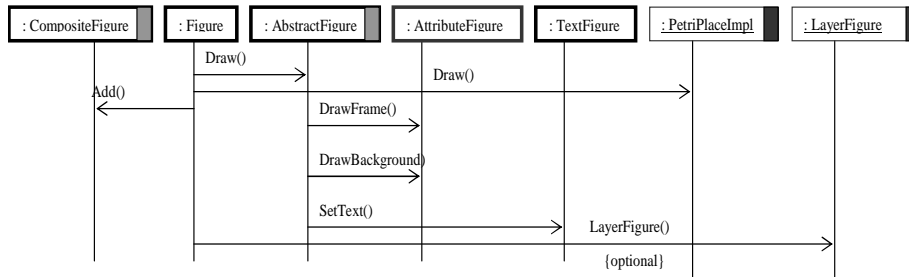


Figure 5: Generated Sequence diagram for “Create Figure”

5 - EVALUATION OF THE FBDM METHOD

The evaluation of FBDM is divided into two phases. In the first phase, the quality of the generated framework is examined with respect to the quality of the original applications; the quality is evaluated using several object-oriented design metrics. In the second phase, the generated framework is compared with the existing framework (JHotDraw) to examine its completeness, consistency and non-redundancy. This comparative evaluation allows us, in addition, to evaluate the usefulness of the F-UML notation.

5.1 - Evaluation Metrics

Many researchers have been interested in software metrics (c.f., (Kim,2002), (Shepperd, 1993)) and particularly in object-oriented specific metrics (c.f., (Chidamber, 1994), (Lorenz, 1994), (Xenos, 2000)). Xenos (2000) presents a state of the art of object-oriented metrics and classifies them essentially into three categories: class, method and inheritance metrics.

Kim (2002) adapted these object-oriented metrics for UML diagrams. In our evaluation, we have selected a set of most pertinent metrics for frameworks from the works of Xenos (2000) and Kim (2002). To these metrics, we added a set of framework specific metrics.

As illustrated in Table 4, the selected metrics for each diagram can be divided into two categories, depending on their usage. The first category of metrics can be used to measure the quality of a diagram in terms of its structure. The second category of metrics (marked with ^R) helps to identify the reuse degree of a diagram, e.g., the number of classes in the core, the number of methods that must be redefined when reusing a whitebox class, etc. We note that the metrics listed in Table 4 can be used to calculate other metrics, e.g., the number of hot-spot use cases (NHU) and the number of hot-spot actors (NHA) can be derived by subtracting NCU from NUM and NCA from NAM, respectively.

5.2 – An Intrinsic Evaluation

In this intrinsic evaluation of the generated framework, we aimed at testing the efficiency of the FBDM process. More precisely, we wanted to answer the following question: starting from “good” quality applications, does the FBDM process generate an “as good” quality framework? For this, we used the set of quality metrics to examine the generated framework.

Table 5 summarizes the values of the metrics used to answer this question. Overall, as for the class diagrams of the applications, similar results were confirmed for the quality of the generated framework class diagram since it is the largest, we next examine it more closely.

Number of classes

The generated class diagram has a fewer classes than the largest application (RENEW). The excluded classes are due to **Rule 6**, which uses the heuristic R_{dc} to decide whether a class covers the domain enough to be added to the generated framework. Recall that R_{dc} determines the importance of a class in the domain as a function of the number of appearances of the class in the unified applications. In other words, this heuristic lets us eliminate the classes that are too application specific. For instance, the class PetriEditor which belongs to the JARP application was omitted in the generated framework, since both $R_{dc}(\text{PetriEditor}) < 2/3$ and this class is far from a core class by more than two inheritance levels.

Number of relations

In addition, the number of relations (association, aggregation and inheritance) slightly increased in the generated class diagram due to **Rule 7** which automatically transfers all relations between classes added to the framework class diagram. For instance, according to the FBDM process, the DIT of the generated framework will be at most one plus the maximum of the DITs of the original applications. The additional depth level results from **Rule 3.d** and justifies the value of NIM for the generated framework. For example, consider the DIT of the class AbstractFigure: in the JOONE

application, it is equal to 4, in the JARP and RENEW applications it is equal to 3, while it is equal to 4 in the generated framework.

Similar results were confirmed for the quality of the generated use case diagram and sequence diagrams.

<i>Metric</i>	<i>Definition</i>	
Class diagram	NCM	Number of classes in a model (Kim, 2002)
	NCC	Number of core classes ^R
	NBBC	Number of blackbox hot-spot classes ^R
	NWBC	Number of whitebox hot-spot classes ^R
	NWBCC	Number of whitebox core classes ^R
	NAsM	Number of associations in a model (Kim, 2002)
	NAgM	Number of aggregations in a model (Kim, 2002)
	NIM	Number of inheritance relations in a model (Kim, 2002)
	<i>Class metrics</i>	
	NAtC	Number of attributes in a class (Kim, 2002)
	NOpC	Number of operations in a class (Kim, 2002)
	NROC	Number of re-definable operations in a class ^R
	<i>Inheritance metrics</i>	
	DIT	Number of ancestors of a class (Chidamber,1994)
NOC	Number of children of a class (Chidamber,1994)	
Use case diagram metrics	NAM	Number of actors in a Model (Kim, 2002)
	NCA	Number of core actors ^R
	NUM	Number of use cases in a Model
	NCU	Number of core use cases ^R
Sequence diagram	NOM	Number of objects in a Model (Kim, 2002)
	NMM	Number of messages in a Model
	NOpM	Number of optional messages in a Model ^R

Table 4. Evaluation metrics used (^R: reuse metric)

Metric	Generated Framework	JOONE	JARP	RENEW
NCM	220	242	175	328
NAsM	93	35	33	191
NAgM	8	48	24	17
NIM	231	132	123	243
NAtC	0 ... 10	0 ... 43	0 ... 16	0 ... 10
NOpC	1 ... 58	1 ... 80	1 ... 31	1 ... 103
DIT	0 ... 17	0 ... 8	0 ... 15	0 ... 17
NOC	0 ... 3	0 ... 5	0 ... 3	0 ... 3
NAM	1	1	1	1
NUM	34	14	13	21
NOM ("CreateFigure")	7	7	4	6
NMM ("CreateFigure")	8	6	4	5

Table 5. Values of quality metrics

Metric	Result
NCC	38
NBBC	76
NWBC	102
NWBCC	4
NROC	0 ... 7
NCA	1
NCU	7
NOpM("CreateFigure")	2

Table 6. Values of reuse metrics

Overall, as for the class quality, the case study illustrates that the generated classes maintain the quality metric values below the largest values of the unified classes.

While there are no fixed thresholds for the above metrics, it is clear that their values reflect the ease of understanding a design and the benefits of reusing it. For example, classes with a large

number of attributes (NAtC) and methods (NOpC) would be of a higher complexity (Chidamber,1994). Another example is the DIT of a class, which measures the number of ancestor classes. The deeper a class is within the hierarchy, the greater the number of methods it inherits and, thus, the more complex the class is. However, while deeper trees yield a greater design complexity, they offer better reuse levels through the multiple inherited methods.

A third example is the metric NOC, which both measures the number of a class children of and indicates its possible reuses: A large number of children implies a greater reuse potential. However, this measure alone may not convey a precise reuse measure. For instance, consider the NOC of the class *AbstractFigure*; it is equal to 3 both in JHotDraw and the generated framework, which indicates that *AbstractFigure* offers a medium potential of reuse. However, in the generated framework, *AbstractFigure* is the root of an *incomplete* hierarchy (see Figure 4), which indicates that the generated framework can be adapted to a particular application by adding other inheriting classes. Thus, with the F-UML notation, the number of incomplete relations stemming from a class can be combined with NOC for a better indication of the reuse measure.

5.3 – A Comparative Evaluation

In this evaluation, we aimed at testing the completeness and correctness of the generated framework, and the usefulness of the F-UML notation.

Consistency and completeness of the use case diagram

Comparing the use case diagram of the generated framework (Figure 3) with the manually designed use case diagram of JHotDraw, we noted that:

- The same main actor (user) is present in both frameworks.
- The 27 use cases representing application-specific functions (e.g., create a neural net figure, create an arc) are generated as hot-spots that illustrate possible adaptations of the framework.
- The set of use cases in the generated framework contains most of those of JHotDraw. More specifically, NUM is 9 in JHotDraw and it is 34 in the generated framework from which 27 are hot-spot use cases (i.e., shown only to give an idea on possible reuses) and 7 are core use cases (see Table 6). Thus, comparing JHotDraw with the generated framework, only two use cases ("create animation" and "use

construction tools for figure") were not detected by the design process. Since they were not used by any of the three applications, these use cases do not represent essential functions of JHotDraw. In addition, relying on the F-UML notation, the missing use cases can be easily added as hot-spots.

Overall, this case study confirmed the completeness and correctness of the unification rules for the use case diagrams, since the JHotDraw use case diagram is contained in the generated use case diagram, modulo the hot-spots.

Consistency and completeness of the class diagram

Comparing the class diagram of the generated framework (Figure 4) with the class diagram of JHotDraw, we noted that:

- All of the abstract classes which define the generic structure and behavior of any application in the framework domain (e.g., Figure, AbstractFigure, CompositeFigure, FigureChangeListener, Drawing) are completely and correctly derived by the FBDM design process.
- The concrete classes that could be reused in specific applications (e.g., EllipseFigure, RectangleFigure) are generated as hot-spot classes.
- The NCM of JHotdraw is 216 while it is 220 in the generated class diagram. Looking closely at both diagrams, we found out that:
 - The additional classes are marked as hot-spots in the generated framework. This may lead us to conclude that the design process produces a framework with (possibly too many) application specific increments, e.g., PetriArc, PetriPlace, etc. These latter could be considered as details that may complicate the comprehension of a framework and hence impede its reuse. However, the F-UML notation helps by visually distinguishing these details from the core. Thus, when reusing a framework, the designer can first focus on the core, and later he/she can choose to understand or ignore the hot-spots.
 - Some classes in JHotDraw were not derived in the generated framework because they were absent in the original applications (e.g., *ImageFigure* in JHotDraw). However, the FBDM design process puts the tag {incomplete} wherever the designer can add the

missing classes. For instance, *RoundRectangleFigure* can be added as an inheriting class of *AttributeFigure* in the generated framework; this possibility is marked by the tag {incomplete} on the generalization out of *AttributeFigure* (see Figure 4). These missing classes may reduce the number of reused classes (as a reuse metric); however, thanks to the F-UML notation, the designer is advised of the places he is expected to focus his design effort.

- The large number of classes in the derived framework is justifiable since the original applications have an average NCM of 248 classes. In addition, a large number of the framework classes is a blackbox hot-spot (NBBC=76); that is, they give an idea about possible framework adaptations thus, one would only select some of these classes without having to modify them.
- The relations (generalization, association and aggregation) derived in the framework are consistent with their counter parts in JHotDraw. The generated framework contains additional relations taken with the application-specific classes.

Similar results were noted when comparing the sequence diagrams of the generated framework with the sequence diagrams of JHotDraw. The generated sequence diagram includes the corresponding sequence diagram in JHotDraw. In addition, it complements it with an optional message (*LayerFigure()*) relating the core object *Figure* and the hot-spot object *LayerFigure*.

Overall, the generated framework is consistent with and contains more details than JHotDraw; In addition, the degree of details produced in the generated framework depends on the level of domain coverage of the unified applications. F-UML helps identifying the details, which is essential both to measure the degree of reuse and to guide a reuse. However, it lacks concepts to express a conditional instantiation of two alternative hot-spots. For example, there is no indication in the class diagram of the framework of Figure 4 that the class *NeuralNetDrawing* cannot be taken with *PetriNetImpl*. This leads us to consider the use of OCL to express such constraints in future work. Finally, the F-UMLTool was vital in the design process, especially in managing the complexity of applying the rules on the large class diagrams.

Finally, we note that in addition to the JHotDraw experiment reported in this paper, we have

evaluated the FBDM method in the case of a framework for e-commerce brokers (Bouassida et al., 2002). This latter case study contains all three diagrams. The framework was generated from three independent e-broker applications. Overall, the e-commerce broker experiment confirms the above reported results for the graphical editor domain. Unlike the JHotDraw case study, we do not have access to an e-broker framework to compare the generality and degree of reuse of the derived framework.

6 - CONCLUSION

This paper first presented the FBDM design method for frameworks. It then presented an experimental (intrinsic and comparative) evaluation of FBDM in the domain of graphical drawing editors.

On one hand, the case study showed that the F-UML notation facilitates the distinction between the core of the framework, which must be present in any application derived from it, and its variable parts. This, in turn, can guide the designer in estimating the degree of reuse the framework can offer. In addition, the case study highlighted the fact that the F-UML notation might need to be augmented with the Object Constraint Language (OCL) to express certain reuse constraints on the hot-spots.

On the other hand, the case study showed that the design process generates a framework that contains the whole framework core, certain hot-spots present in JHotDraw, and other application-specific hot-spots.

We are currently working on three research axes. The first examines how to add the generation of the pattern diagram to the F-UMLTool. The second consists of automating the collection of the semantic comparison criteria in the dictionary used by the F-UMLTool during the design process. Finally, our third research axis examines how F-UML can be integrated in the OMG Model Driven Architecture (OMG, 2001); in particular, we are examining how to define the PIM-PIM and the PIM-PDM transformations in terms of hot-spots. The integration of F-UML in MDA provides MDA with a reuse validation capability through the formal semantics of F-UML (Bouassida et al., 2003) (Bouassida et al., 2005).

BIBLIOGRAPHY

- Aksit, M., Tekinerdogan, B., Marcelloni, F., Bergmans, L. (1999), "Deriving Object-Oriented Frameworks from Domain Knowledge", in *Building Application Frameworks: Object-Oriented*

- Foundations of Framework Design*, M. Fayad, D. Schmidt, R. Johnson (Eds.), John Wiley & Sons Inc., pp169-198.
- Bouassida, N., Ben-Abdallah, H., Gargouri, F., Ben-Hamadou, A. (2004), "F-UMLTool for the formal design of frameworks", *XXII^{me} Congrès INFORSID*, Biarritz-France 25-28 Mai.
- Bouassida, N., Ben-Abdallah, H., Gargouri, F., Ben-Hamadou, A. (2002), "A stepwise Framework Design Process", *IEEE International Conference on Systems Man and Cybernetics*, Hammamet, Tunisia, 07-09 October.
- Bouassida N., Ayadi, T., Ben-Abdallah, H., Gargouri, F. (2002), "Design of a framework for electronic commerce brokers", *IEEE International Conference on Cognitive Informatics*, Calgary, Canada, 27-29 August .
- Bouassida, N., Ben-Abdallah H., Gargouri F., Ben Hamadou A. (2003), "Formalizing the framework design language F-UML", *International conference on Software Engineering and Formal methods (SEFM'03)*, Brisbane-Australia.
- Bouassida, N., Ben-Abdallah, H., Gargouri, F., Ben-Hamadou, A. (2005), "Towards a rigorous architectural reuse", *Arab International conference on computers Software and Applications*, Egypt.
- Chidamber, S. R., Kemerer, C. F. (1994), "A metrics suite for object-oriented design", *IEEE Transactions on Software Engineering*, Vol 20, N° 6, pp 476-493.
- Erni, K., Lewrentz, C. (1996), "Applying design metrics to object-oriented frameworks", *Proceedings of the 3rd International Software Metrics Symposium (Metrics'96)*.
- Fayad, M., Schmidt, D., Johnson, R. (1998), *Building Application Frameworks*, Wiley.
- Fontoura, M.F., Pree W., Rumpe B. (2000), "UML-F: A Modeling Language for Object-Oriented Frameworks", *European Conference on Object Oriented Programming*, Springer-Verlag.
- Fontoura, M. F., Crespo S., Lucena C.J., Alencar P., Cowan D. (2000), "Using viewpoints to derive Object-Oriented Frameworks: A case study in the web education domain", *The Journal of Systems and Software (JSS)*, vol 54, n°3 Elsevier Science.
- Gamma, E., Helm, R. Johnson, Vlissides, J. (1995), *Design patterns: Elements of reusable Object Oriented Software*, Addison-Wesley, Reading, MA.
- Gamma, E., Eggenschwiler, T. (2000), <http://www.jhotdraw.org>
- Johnson, R. E., Foote, B. (1998), "Designing reusable classes", *Journal of Object Oriented Programming*, vol. 1, n°2.
- Kim, H., Boldyreff C. (2002), "Developing Software Metrics Applicable to UML Models", *Workshop QAOOSE, Malaga-Spain*.
- Koskimies, K., Mossenback, H. (1995), "Designing a framework by stepwise generalization", *5th European software Engineering Conference., Lecture Notes in Computer Science 989*, Springer-Verlag.
- Lorenz, M., Kidd, J. (1994), *Object-Oriented Software Metrics: A Practical Approach*, Prentice Hall.
- Meyer, B., (1988) *Object Oriented software construction*, Edition Prentice-Hall International.
- OMG. Model-Driven Architecture Home Page, <http://www.omg.org/mda> (2001).
- Pree, W. (1994), "Meta-patterns: a means for capturing the essentials of object-oriented designs", *European Conference on Object Oriented Programming*, Bologna, Italy.
- Rational, www.rational.com, 2004.

- Riehle, D., Gross, T. (1988) "Role model based framework design and integration", Proceedings of OOPSLA'98, Vancouver.
- Reenskaug, T., (1996), *Working with objects*, Greenwich : Manning.
- Roberts, D., Johnson, R. (1996), "Evolving Frameworks: A pattern language for Developing Object Oriented Frameworks", *Proceedings of the third conference on pattern languages and programming*, Montecilio, Illinois.
- Rumbaugh, J., (1991), *Object Oriented Modelling and design*, Prentice Hall.
- Sanada, Y., Adams, R. (2002), "Representing Design Patterns and Frameworks in UML-Towards a Comprehensive Approach", *Journal of Object Technology*, vol. 1, n°2, July-August.
- Shepperd, M.J., Ince, D. (1993), *Derivation and Validation of software Metrics*, Clarendon Press, Oxford, UK.
- JARP (2001), Petri Nets Analyzer: JARP, SourceForge, <http://www.jarp.org>.
- JOONE (2001), Java Object Oriented Neural Engine: Joone, SourceForge, <http://www.joone.org>.
- Schmid, H. A. (1997), "Systematic framework design by generalization", *Communications of the ACM, Special issue on Object Oriented Application frameworks*, Vol 40, N°10.
- Smith, G. (2000), *The object-Z specification Language*, Advances in Formal methods, Kluwer Academic Publishers.
- Szyperski, C., Pfister, C. (1996), Workshop on Component Oriented Programming in Mülhäuser M. (editions), *Special issue on Object Oriented Programming, ECOOP (96)*, Vrelag, Heideberg.
- Renew, (2001) Wienberg, F., Kummer, O., Duvigneau M., <http://www.renew.org>.
- Xenos, M., Starvrinoudis, D., Zikouli, K., Christoudalis, D. (2000), "Object-Oriented Metrics- A Survey", *Proceedings of the Federation of European Software Measurement Associations*, Madrid, Spain.