Evaluating Composition Techniques for Architectural Specifications: A Comparative Study

Ana Luisa Medeiros
OPUS Research Group,
Computing Department,
Federal University of Rio Grande do Norte (UFRN)
analuisa@ppgsc.ufrn.br

Kleinner Farias, Alessandro Garcia
OPUS Research Group,
Department of Informatics,
Pontifical Catholic University of Rio de Janeiro (PUC-Rio),
{afgarcia,kfarias}@inf.puc-rio.br

Thais Batista
Software Engineering Team,
Computing Department,
Federal University of Rio Grande do Norte (UFRN),
thais@ufrn.br

ABSTRACT

Techniques for composing architectural specifications are emerging in order to facilitate software architecture evolution. However, there is little empirical understanding on whether such techniques scale when they are used to express different types of architectural changes. This paper presents a first comparative evaluation of two significantly-different composition techniques for architectural descriptions. The first technique is fully based on heuristic composition operators, while the second one demands explicit composition specification. Several releases of a software product line were used in our evaluation, and their designs were expressed with an architectural description language, called ACME. Some metrics were used to compute the number of required modifications, syntactic conflicts, and semantic conflicts in composed (output) models produced with both heuristic and non-heuristic compositions. We observed that, in general, the heuristic composition approach outperformed the non-heuristic one mainly due to the narrow set of composition operators supported by the latter.

Keywords

Software Architecture, ADLs, Metrics, Assessment, Model Composition.

1. INTRODUCTION

Techniques for explicitly describing the composition of architectural specifications are emerging [2] [12] [13]. These techniques allow handling relations and compositions of architectural models by embedding their specifications in architectural description languages (ADLs). They can be used for different purposes, such as reconciling architectural descriptions developed in parallel by different development teams, or evolving architectural descriptions with models of new features being added to the architecture. However, such explicit compositional ADL-based approaches have rarely been assessed and they have not proven their value yet. For instance, it is not clear to what extent these techniques scale when they are used to express different types of architectural changes. They have not even compared with general well-known heuristic approaches for model composition [5] [6].

In this context, this paper presents an exploratory comparison of two techniques for composing architectural descriptions: (i) a conventional heuristic approach [1] [5] [6], where input models are automatically composed and output models are generated based on pre-defined algorithms, and (ii) a non-heuristic approach [2], where the compositions of two input models are fully described using a limited set of composition operators. The architectural models are represented with either textual or graphical notations. We evaluate these techniques in the context of an evolving product-line architecture. The compositions of architectural descriptions are used to represent the evolutions of the target product-line software architecture.

The models of the target product line were based on the ACME ADL [3]. This language was selected because, unlike most ADLs, it is not domain-specific and provides generic structures to describe a wide range of systems [3]. ACME is supported by tools that provide a good basis for designing and manipulating architectural descriptions and generating code. The key elements of ACME are components, ports, roles, connectors, bindings and attachments. *Components* are potentially composite computational encapsulations that support multiple interfaces known as *ports*. Ports are bound to ports on other components using first-class intermediaries, called *connectors*, which consist of one or more interfaces, called *roles*. Roles are directly attached to component ports via *attachments*. The attachments section (configuration) defines a set of port/role associations. The *bindings* provide a way of associating a port with a component.

In our study, we have also used a suite of four metrics to analyze the quality of the composed models produced by the techniques that are being assessed. First, we have used three metrics [1] to compute the rate of syntactic and semantic conflicts in the composed architectural descriptions. These metrics were used to quantify [1]: (i) the number of *semantic clash conflicts*, (ii) the number of *behavioral feature conflicts*, and (iii) the number of *unmeaning model elements*. Furthermore, we also measure the number of changes required in the output model in order to resolve the identified conflicts (and other design anomalies). This change computation is important because it allow us to assess which composition technique requires less effort to realize the intended composed model.

The paper is structured as follows. Section 2 includes a background about model composition and the techniques being assessed. Section 3 presents the methodology used to compare the heuristic and non-heuristic approaches. Section 4 reports the results and their analyses. Section 5 presents related work and

threats to validity. Section 6 introduces some final remarks and directions for future work.

2. BACKGROUND

This section presents the main concepts that are used throughout the paper and which are fundamental to understand the results of our comparative analysis.

2.1 Model Composition in a Nutshell

The term *model composition* can be defined as a set of activities that should be accomplished to combine two (or more) input models, M_A and M_B, in order to produce an output composite model, M_{AB}. Model composition techniques have a built-in model composition strategy (e.g., override and merge) that are responsible for defining the semantics of a model composition relationship and specifying how the input models should be manipulated in order to compose them. The composition techniques can be either heuristic-based approaches, which rely on "guessing" the semantics of model elements, or non-heuristic-based approaches, which requires that modelers interpret the semantics of the input model elements during the model composition process. Non-heuristic-based approaches aim at, in general, being semantic-preserving, meaning that modelers should use the composition operators only if it is possible to preserve the semantics of the existing elements in the input models. Each technique produces an output model that can be differentiated by the terms *composed model* (M_{CM}) and *intended model* (M_{AB}) (or *ideal model*) to refer to the composition produced by an composition algorithm and the composition that the developer desires, respectively.

The main difference is that the composed model can have some model composition conflicts [1] [4] e.g., semantic clash conflicts, behavioral conflicts. Thus, to overcome these conflicts, the designer needs to make some modifications into the composed model in order to produce the intended model. We can basically have two types of composition conflicts [1] [4]: (1) syntactic conflicts, which arise when the composition algorithm results in a model not conforming to the modeling language's metamodel; and (2) semantic conflicts, where the meaning of the composed model does not match that of the intended model.

2.2 Heuristic Approach

The heuristic approach is based on two key activities: matching model elements in the input models, and composing elements in order to generate the output models. This paper focuses on two well-established composition algorithms: *override* and *merge* in the context of the architectural level. These algorithms were chosen because they have been applied in a wide range of model composition scenarios, such as model evolution, ontology merge, and conceptual model composition. In addition, they have been recognized as candidate algorithms in aspect-oriented model composition (e.g., Theme/UML [5] [6]) and UML diagrams [8].

In the following, we briefly define these three algorithms, where we assume two hypothetical input architectural models (systems), A and B. We say that two elements from A and B respectively are *corresponding* if they have been identified as equivalent in the matching process. Matching can be achieved using any number of standard heuristics, such as match-by-name [1] [4] [9].

Override (direction: A to B). For all pairs of the corresponding architectural elements (components, roles, ports, bindings, connectors and attachments) in A and B, Model A's architectural elements should override Model B's corresponding architectural element. Architectural elements not involved in the correspondence remain unchanged and are inserted into the output model.

Merge. For all corresponding architectural elements in A and B, the elements should be combined. The combination depends on the architectural element type. In this paper, we consider components, ports, connectors, attachments and bindings – in this case, the combination is to add the ports of A's element to those of B. Architectural elements in A and B that are not involved in a correspondence match remain unchanged and are inserted into the output model directly.

2.3 Non-Heuristic Approach

In a non-heuristic approach, the results of the composition are originated from textual or graphical notations. No heuristic is used to automatically realize the composition between models. In [2] three strategies of compositions are introduced and incorporated to xADL [7]. These relations are summarized in the following definitions.

Unification. Elements are unified from several structural views and the element to be unified needs the same element. After the composition, the elements must have the same set of ports or the software architecture needs to define the corresponding interfaces or ports before composition. The unified elements can be components, connectors, and bindings defined in multiple structural views of the software architecture. A unified element in the output (composed) model represents simultaneously those elements that appear in different views (input models).

Mapping. Individual elements or groups of elements called subjects from one structural view are mapped to on a single element of another structural view that are called target element. The subjects elements are elements chosen to become subelements of a target element in the composed (output) model. The points on which the elements are connected to each other are the interfaces or ports. These composition points are called joinpoints. The corresponding interfaces or ports between the target and the subject must be to define by architect.

Refinement. A specific structural view (referred to as inner structure) describes a substructure for an architectural component (referred to as outer component) of another structural view. The architect needs to specify corresponding interfaces or ports (joinpoints) between the outer component and interfaces in the inner structure.

Figure 1 illustrates an example using the refinement strategy. It shows three structures (represented by `Structure1`, `Structure2` and `Structure3`). Applying the refinement, the `Structure2` refines ComponentA of `Structure1`, resulting in `Structure3`. After composition, the `Structure2` components are subelements of ComponentA. The joinpoints (specification of relationship) between the structures are the interfaces. For example, the `interfaceup` interface of `ComponentA` is mapped on interface `up` of its subcomponent Z.

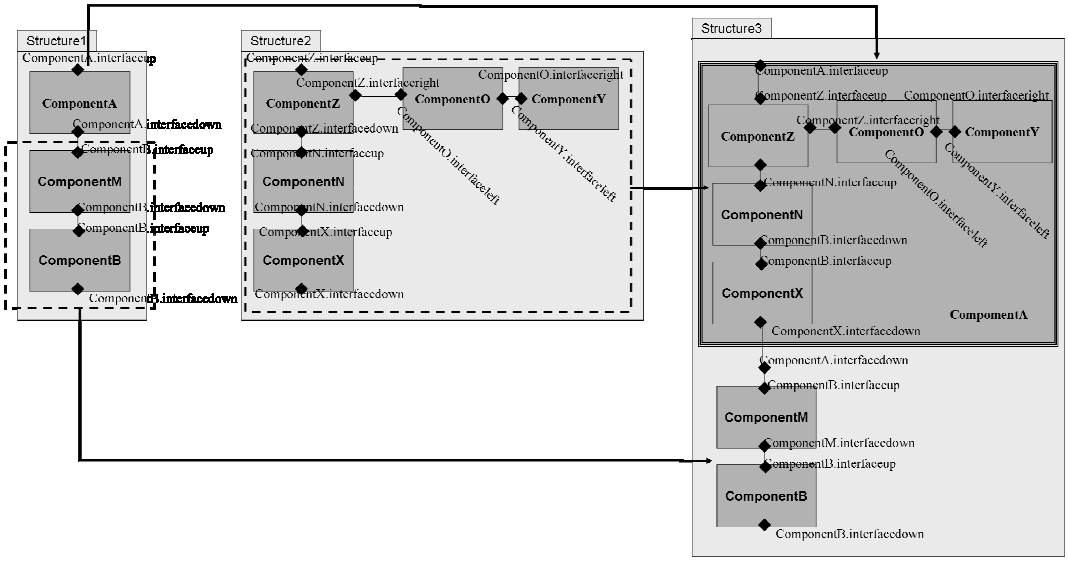

Figure 1. Example of Strategy Refinement with xADL

3. EVALUATION METHODOLOGY

We have carried out an *exploratory* comparison of heuristic and non-heuristic techniques for composing architectural descriptions. There was no sort of control of variables in our study. The goal of this initial (pilot) assessment was to have a first understanding on the usefulness of alternative composition techniques for allowing us to express the evolution of ADL-based architectural descriptions. With the outcomes of this comparison, we hope to be able to start to derive some hypotheses on when/how to better use such alternative composition techniques, and organize more controlled experiments in the near future.

The following subsections describe the basic evaluation steps carried out in our exploratory evaluation, including: (i) the selection of the target application for comparing the use of heuristic and non-heuristic architectural compositions (Section 3.1), (ii) the procedures to produce the composed architectural descriptions with both architectural composition techniques (Section 3.2), and (iii) the metrics used to evaluate the quality of the output composed descriptions (Section 3.3).

3.1 Target Product Line

A software product line (SPL), called Mobile Media (MM) [11], was selected as the target application in our comparative study. We have chosen a product line because it is often natural [8] to: (i) decompose the architectural elements that realize each SPL feature in a separate architectural description (view), and (ii) then, compose them with the base architecture (and other features) to produce the specific product architecture descriptions. Mobile Media supports the handling of photo, video, and music data on mobile devices, such as mobile phones. This SPL was chosen because its architecture has undergone different types of evolutions, including additions of new features, change of a mandatory to an alternative feature, and so forth [11].

The Mobile Media Architecture Descriptions. The Mobile Media system was also interesting because each change set (for each release available) was properly documented. All MM evolutions have been recorded in an evolution description document that is used to create the delta model [12], M_B (Section 2.1), i.e., the description that specifies the new architectural changes to be accommodated into the base architecture description, M_A. Therefore, the architectural MM artifacts were instrumental to allow the investigation of potential problems on the use of composition strategies (Section 2) for expressing evolving architectural models of the SPL. In addition, the architectural specifications were not fully complete and had some ambiguities too, which was interesting as well. These imperfections mimic how architectural descriptions are produced in real-life projects. The original designers of the MM system were available to address emerging concerns during both our pre-study procedures and on the evaluation of the analysis results.

Figure 2 illustrates part of the MM architecture using the ACME graphical notation. Furthermore, this figure represents one part of the Mobile Media before composition and after application of the refinement strategy (Section 2.3). Eight implementations (and the respective architectural designs) of MM releases were available in Java programming languages [9-11]. For the illustrated part of the MM architecture, we will mainly use two components to illustrate the results and discussions in this paper: (i) the `MediaControl` component provides services to media management, and (ii) the `MediaAdditiontoAlbum` that requires services to include an additional type of media in the screen of the `Album`.

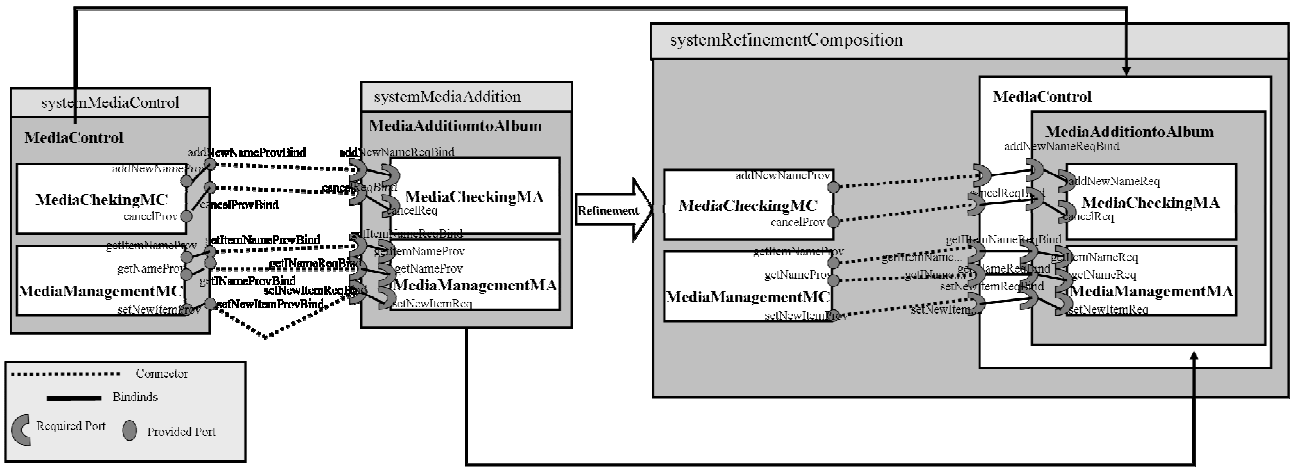

Figure 2: ACME Representation of some Mobile Media components using Refinement Strategy

3.2 Producing the Architecture Compositions

For each release of MobileMedia, we have applied each of the composition techniques described in Section 2. That is, we have used the heuristic strategies (Section 2.2) to compose two input architectural descriptions in order to produce a new release description of the target SPL (Section 3.1). While one of the inputs represented the current MM architecture, the other input represented (for instance) the architectural elements of the new feature being added. Similarly, we used the non-heuristic technique to fully specify the compositions of the two input descriptions and, then, generate the next release of the MM architecture descriptions. The goal was to identify if the quality of the output models, in terms of the metrics selected (Section 3.3), were the same or different for the compositing techniques being compared.

Example of Non-Heuristic Composition. Figure 2 represents what happens with components and their relationships when the refinement strategy (Section 2.3) is applied. We can observe that bindings are removed. A number of bindings are affected by the composition: (i) `cancelProvBinding` and `cancelProv` ports between `MediaControl` and `MediaCheckingMC` components; (ii) `addNewNameProvBinding` and `addNewNameProv` components; (iii) `getItemNameProvBinding` and `getItemNameProv` ports between `MediaControl` and `MediaManagementMC` components; (iv) `getNameProv Binding` and `setNewItemReqBinding` between `Media Control` and `MediaManagementMC` components. A number of roles involved in component connections are also modified. This happens because we choose the refinement strategy. Therefore, we needed to have another structural architectural view centralizing components with similar functions. Hence, the following process has been applied. The `MediaAdditiontoAlbumScreen` component and its subcomponents are transformed in the inner structure of the `MediaControl` component. So, bindings are created or adapted to connect the `MediaCheckingMC` and `MediaManagementMC` components with the `MediaCheckingMA` and `MediaManagementMA` components and existing ports before of the application of the refinement strategy in `MediaControl` of `systemMediaControl`.

Example of Heuristic Composition. Figure 3 shows the composition using the heuristic merge strategy. MM has two 'subsystems': (i) `systemMobileMediaA` encompasses the specification of the MobileMedia part that is responsible for controlling all the different types of media – this system is mainly formed by the architectural components `MediaControl` and `MediaAdditiontoAlbum`; and (ii) `systemMobileMediaB` encompasses the additional architectural components with general functionalities, such as control media, add media, capture media, and so forth. The goal is to generate an output system that has `MediaControl`, `MediaAdditiontoAlbum` and `Media Capturing` components including ports that required and provided new services. For example, the `getCaptureMediaProv` and `setCaptureMediaProv` ports added to `MediaControl` component and respective bindings to connect `MediaManagementMCa` and `MediaCapturing`. The Merge strategy was chosen in this case because the change scenario (in this release) involved the addition of new architectural elements to the model. If the override strategy was applied, this could imply in some elements being undesirably deleted (i.e. overriden).

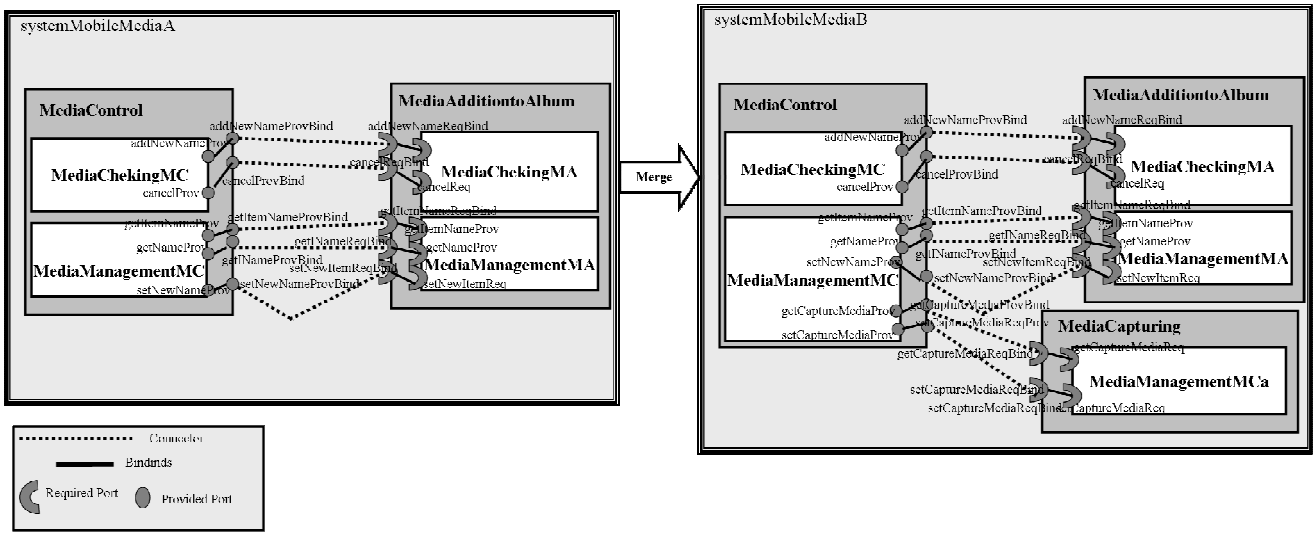

Figure 3. Partial ACME Representation of Mobile Media using Merge Strategy

3.3 Measuring the Architecture Compositions

This section presents the metrics used in our study in order to quantify: (i) the modifications required to fix the composed models (Section 3.3.1), and (ii) the composition conflicts arising in the composed architectural descriptions (Section 3.3.2). These metrics are particularly interesting because they work as indicators of the imperfections present in the composed (output) architectural descriptions. Therefore, they somehow indicate the effort required by the architects to transform the output description into the intended resulting architecture. The used metrics were originally defined and validated in previous studies [1] [4]. No major adaptation was required in order to apply these metrics in the composed architectural models specified in ACME.

3.3.1 Quantifying Description Modifications

Number of Modifications of the Elements (NME). This metric counts the *number of modifications of the elements* in the architectural description. The number of modifications is an indicator of what approach produces a model that requires the architect to spend more time to fix the composed description. It is likely the case that a description with higher number of required modifications also contains more syntactic and semantic conflicts to be resolved (Section 3.3.2).

NME = /M/, where *M is the number of modifications of the elements in a model after the composition process.*

3.3.2 Quantifying Composition Conflicts

Number of Semantic Clash Conflicts (NSCC). NSCC is used to calculate the number of semantic clash conflicts in a model. A semantic clash conflict occurs when model elements have different names, however, with the same semantics [1] [4]. For example, a port of the component `MediaManagementMC` (Figure 2), named `setNewNameProv`, is responsible for adding the name of the item related the Media; the presence of this port might conflict with another port named `addNewNameProv` of the `MediaCheckingMC` component which implements exactly the same functionality. This metric is particularly interesting as its results enable us to understand the rate of undesirable semantic redundancy of an output model. It may also affect the output model understandability and, in turn, complicate the 'recovery' of the generated architecture (with respect to the intended composition). This metric is given by the following formula [1]:

NSCC = /M/, where M is the number of Semantic Clash Conflicts.

Number of Behavioral Feature Conflicts (NBFC). NBFC counts the number of behavioral conflicts emerging in an architectural component specification. A behavioral feature conflict may occur when a component: (1) has two (or more) ports that are used with the same purpose; for example, when an `MediaManagementMC` component that has a port named `getItemName` and other port named `getName` (see Figure 2), and (2) refers to a port that no longer exists, or exists under a different behavior (connection) that is not expected; for example, when a port requires a service but there is no port that provide the required service. This case can occur in the output model (Figure 2) if it was generated using the override strategy. The `getCaptureMediaProv` and `setCaptureMediaReq` ports would not exist in the model because `ManagementMediaCM` of the `MediaControl` component of the `systemMobileMediaB` would be subscripted by the `ManagementMediaCM` of `MediaControl` component of the `systemMobileMediaB`. So, in the output model the `getCaptureMediaProv` and `setCaptureMediaProv ports` do not exist to provide services to the `getCaptureMediaReq` and `setCaptureMediaReq` of the `MediaManagementMCa` component of the `systemMobileMediaA`. This metric is determined by the formula [1]:

NBFC = /B/, where B is the number of Behavioral Feature Conflicts

Number of Unmeaning Model Elements (NUME). This metric is applied to calculate the number of unmeaning model elements in a description [1]. During the composition process, the model elements are manipulated and sometimes they are artificially isolated from the rest of the architecture specifications.

Unmeaning elements are architectural components, ports, bindings and connections that, after the composition process, become elements with no meaning or purpose. This metric is given by the formula:

NUME= /U/, where U is the number of Unmeaning Model Elements

4. COMPARATIVE ANALYSIS

This section presents the results of applying the conflict metrics and counting the number of modifications (Section 3.3) to reach the intended model during each evolution scenario of the MobileMedia architecture (Section 3.1). Histograms are used to provide an overview of the data collected in the measurement process. These histograms allow us to make a fine-grained comparative analysis of the composition techniques and derive some observations of our comparison. In addition, each histogram is dedicated to a particular composition technique (Section 2). The Y-axis presents the values gathered for a particular metrics. The X-axis specifies the evolution scenarios.

Figure 4 (a) and (b) illustrate the data produced by the conflict metrics (the Number of Semantic Clash Conflicts (NSCC), the Number of Behavioral Feature Conflicts (NBFC), and the Number of Unmeaning Model Elements (NUME)), when applied to the evolution scenario 2 [from R5 to R6]. We focus our attention on this scenario because (1) it is richer than other change scenarios in terms of number of modifications, (2) we observed that some conflicts happened where they would not be expected, and (3) this scenario presents a high number of information to realize the compositions when compared to the other scenarios. This scenario included other architecture capabilities to handle photo and music media. In the other scenarios, there are no significant differences between the values obtained for the two approaches. So we considered this scenario an interesting source of information for our comparative analysis. However, even though the differences were not significant, it is important to point out that the collected data in the scenario 1 [from R4 to R5] and 3 [from R6 to R7] also confirm the conclusions derived from the scenario 2.

Surprisingly, we noticed that the non-heuristic approach showed a higher number of conflicts when compared with the heuristic approach, as would be expected otherwise (see Figure 4). The problem is that the non-heuristic composition strategies are not able to accommodate the required changes correctly – i.e., without giving arise semantic, unmeaning or behavioral conflicts – into the output model. To better understand this statement we will concentrate our discussion on the example depicted in Figure 2, where the merge strategy presents better results for the override strategy (heuristic) than the combined use of non-heuristic operators, i.e. the mapping, unification and refinement.

Figure 4 (a) illustrates the results obtained by applying the conflict metrics to composed models, generated by the heuristic operators in scenario 2 [from R5 to R6]. We can observe, based on the values of NSCC, NBFC and NUME metrics, that the heuristic approach worked better following the merge strategy than the override strategy. The collected data shows that these values occur because in the composition with the merge strategy no subscription of elements appeared as in the strategy override; in the merge strategy, such elements are combined. For example, in Figure 3 (Section 3), we can observe `MediaControl`, `AdditionMediaToAlbumScreen` in the `systemMobileMediaA`. In `systemMobileMediaB`, beyond these components, it also has the `MediaCapturing` component. Hence, the composed model will have a combination of components of the `systemMobileMediaA` and `systemMobileMediaB`, including their inner ports, bindings, connectors, attachments, and subcomponents. In particular, the composed model will have the `setCaptureMediaProv` and `getCaptureMediaProv` ports of the `MediaManagementCM` components, bindings realized between the `setCaptureMediaProvBinding` port of `MediaControl` and `setCaptureMediaProv` port of the `MediaManagementCM`, Furthermore, these ports were created to support the connection between components architectural by ports (see Figure 2) in the output model.

This scenario does not occur in the composition with the override strategy. In this case, the output model would contain information of the `systemMobileMediaA` and `setCaptureMediaProv` and `getCaptureMediaProv` ports of the `MediaManagementMC` components not would be added because `MediaControl` of `systemMobileMediaA` would subscribe `MediaControl` of `systemMobileMediaB`. Hence, we observe that this model, when compared to the intended model, presents more semantic conflicts. For example, the existing connector between `MediaManagementMC` of the `MediaControl` and `MediaManagementMCa` of the `MediaCapturing` would not be in the output model; the same problem applies to the respective bindings and ports. So, the use of the override operator would generate more inconsistencies, lost information, and higher amount of modifications.

When the generated model using override strategy is compared with the intended model the behavioral conflicts arise. The main reason is that there are references in some connectors to a port that does not exist. This fact is also taken into account in the NUME measure. Note that bindings and connections between the ports of the `ManagementMediaMC` and `ManagementMediaMCa` components are unmeaning. Hence the number of modifications using the override strategy in the composed model is bigger than using the merge strategy (see Figure 4 (a)). This situation can be observed in Figure 3, where we would modify the composed model adding two ports referenced in the connector cited previously in the case of the override strategy.

Figure 5 (a), (b) and (c) shows the values of the NSCC, NBFC and NUME metrics collected from the composed model produced with the non-heuristic approach. The NUME measures are equal to zero because descriptions of each strategy are correctly matched. On the other hand, the NSCC and NBFC values in the refinement strategy are higher than in the mapping/unification strategy. This happens because similar elements are not removed when the refinement strategy is used. For example, in Figure 2 we can see that bindings and connections were removed and reallocated. When this model is compared with the intended model, two conflicts are detected: (i) the connection between `addNewNameProv` and `addNewNameReq` ports, and (ii) the binding between the `addNewNameProv` port of the `ManagementMediaCM` subcomponent and the `addNewName ProvBind` port. Hence, when we use the refinement strategy in the composition process, the number of changes in the composed model is higher than when the mapping/unification strategy is used (as shown in Figure 5 (b)). This example helps to explain the

higher NCSS and NBFC values of refinement and mapping/unification when compared with the merge and override heuristics. Furthermore, using refinement or mapping/unification, a higher number of removals and reallocations of the elements is realized.

Considering the number of modifications required to reach the intended model, we observed that the heuristic approach presented a lower measure than the non-heuristic approach in all evolution scenarios (Figure 5 (c)). Note that this measure is obtained by the sum of the number of modifications performed in the scenarios 1 (from R4 to R5), 2 (from R5 to R6) and 3 (from R6 to R7). Moreover, we also observed that the number of modifications of the non-heuristic approach is higher than the results collected from the composition produced following both merge and override strategies.

In Figure 5 (a), we observed that the modifications in the application of refinement, mapping and unification strategies of non-heuristic approach are higher than those resulting of the application of the merge and override heuristics. Based on the data gathered and our experience in previous work [1] [4], this outcome can be explained by the fact that the non-heuristic approach did not preserve the semantic values of the model element.

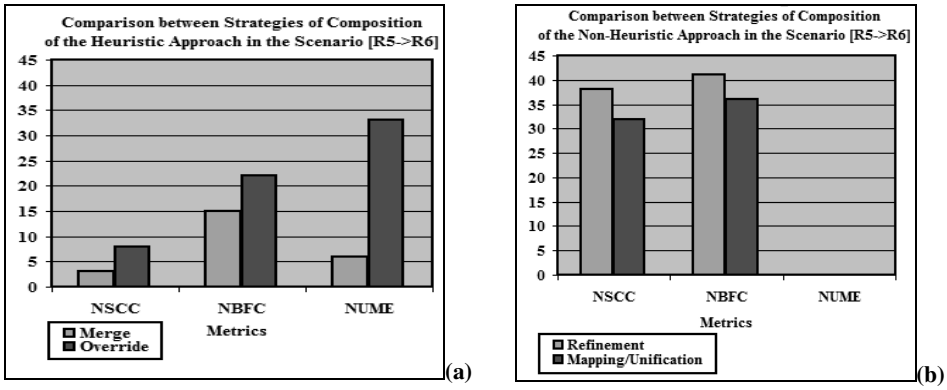

Figure 4. (a) Comparation between Strategies of Composition of the Heuristic Approach in the Scenario [R5->R6] ;(b) Comparation between Strategies of Composition of the Non-Heuristic Approach in the Scenario [R5->R6].

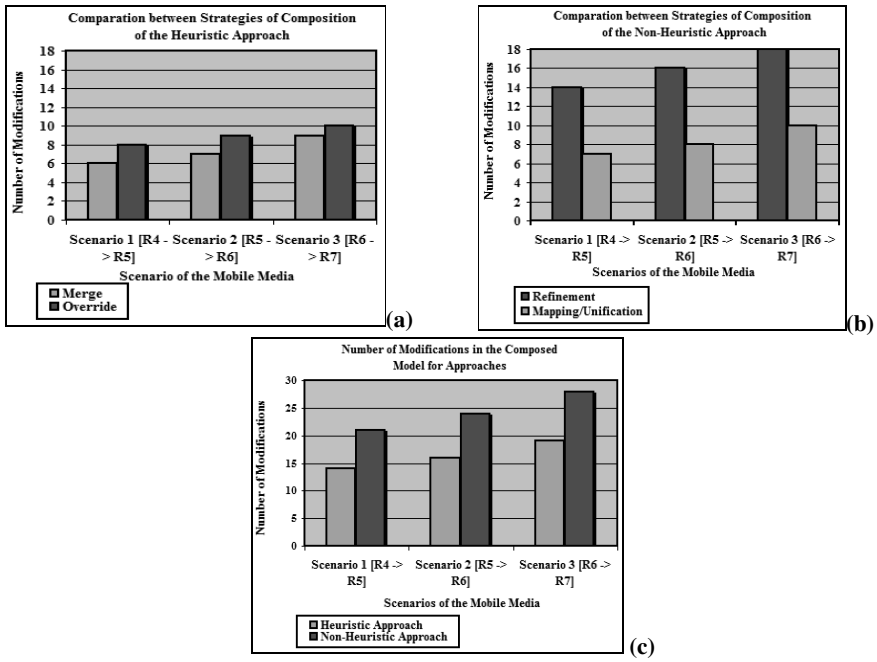

Figure 5. (a) Comparison between Strategies of Composition of the Non-Heuristic Approach for Modifications; (b) Comparison between Strategies of Composition of the Heuristic approach for Modifications; (c) Number of Modifications in the Releases of the Mobile Media for the two Approaches.

5. RELATED WORK AND THREATS TO VALIDITY

Oliveira et al [1] presented a quantitative assessment on applying conventional heuristics (Section 2.2) for composing class models. However, they did not compare the effectiveness of heuristic and non-heuristic techniques for architectural descriptions. Furthermore, all studies related to model composition, including [1], [4], [8], [9], are based on the use of heuristic to realize the compositions. Boucké [2] proposes non-heuristic composition strategies (Refinement, Mapping and Unification) and applies them to architecture descriptions (in several case studies) based on the xADL architectural description language. In addition, the number of repetitions and modifications in the output models of these cases studies were computed to assess the effectiveness of their non-heuristic approach. Different from their work, we compare their non-heuristic approach with a set of model composition heuristics and applied a different suite of metrics.

However, in our study some threats to validity can be obviously identified, including: (i) imperfections in the architectural description compositions as they are not realized in an automatic fashion; (ii) we did not consider the partial results of important metrics, such as time spent and effort to produce the compositions because these partial results were initially obtain and validity by few people; in fact, there is a inherent complexity in the use of the composition strategies, mainly in the non-heuristic approach, because architects need to specify a number of details (e.g. exhaustive specification of relations; (iii) we need to adapt the definitions of some strategies in the non-heuristic approach because they did not originally address certain elements supported by ACME; (iv) the manual application of the metrics of conflicts and number of modifications; and (v) the lack of best practice guidance to apply the heuristic and non-heuristic composition strategies.

6. CONCLUSION AND FUTURE WORK

We presented a comparative study of composition techniques for architectural descriptions. Our initial evaluation has demonstrated that both heuristic and non-heuristic approaches have different deficiencies related to the lack of support for bindings and connections. These deficiencies tend to require more modifications in the composed model by software architects. In addition, we observed that the heuristic merge generated fewer conflicts than the override strategy. For the non-heuristic approach, the results show that the refinement strategy in less efficient because it consistently causes different forms of semantic conflicts. So, more changes are need when using this strategy compared to the mapping/unification strategy. Finally, in general, the heuristic approach outperformed the non-heuristic approach in terms of conflicts and modifications required.

As a direction to future work, we intend to perform the same study with more realistic models. The goal is to compare the results, found in this study, with those collected from the evaluation of a real-world model composition. We believe that, from this comparison, is possible to check whether our initial conclusions can be confirmed or not. Furthermore, we observed throughout the study that we need a new study to investigate whether the metrics used are appropriate for their purpose or not. Depending on the results, we evaluate the necessity of their adaptation, use of other metrics or creation of new ones. Finally, we concluded that new empirical studies are needed to make composition an industrial reality.

7. REFERENCES

[1] Oliveira, K., Garcia, A. and Whittle, J. On the Quantitative Assessment of Class Model Compositions: An Exploratory Study. In: 1th ESMDE at MODELS, 2008.

[2] Boucké, N. Composition and relations of architectural models supported by an architectural description language. PhD Thesis, Katholieke Universiteite Leuven, October 2009.

[3] Garlan, D. Et. al. 1997. ACME: An Architecture Description Interchange Language. In: Proceedings of CASCON '97, 1997.

[4] Farias, K., Garcia, A. and Whittle, J. Assessing the Impact of Aspects on Model Composition Effort. In: AOSD´10, France, March 2010 (to appear).

[5] Clarke, S. Composition of object-oriented software design models, Ph.D. Thesis, Dublin City University, January, 2001.

[6] Clarke, S. and Walker, R. Generic Aspect-Oriented Design with Theme/UML, Aspect- Oriented Software Development, pages 425–458. Addison-Wesley, Boston, 2005.

[7] E. Dashofy, A. van der Hoek, and R. Taylor. A comprehensive approach for the development of modular software architecture description languages. ACM Transactions on Software Engineering and Methodology (TOSEM), 14(2):199–245, 2005.

[8] Oliveira, K., Breitman, K. and Oliveira, T. A Flexible Strategy-Based Model Comparison Approach: Bridging the Syntactic and Semantic Gap. JUCS, vol. 15, no. 11, 2009.

[9] Oliveira, K., and Oliveira, T. Model Comparison – A Strategy-Based Approach, In 20th SEKE, pp 912-917, San Francisco, 2008.

[10] Dashofy, E and Hoek, A, and R. Taylor. A comprehensive approach for the development of modular software architecture description languages. ACM TOSEM, 14(2):199-245, 2005.

[11] Figueiredo et. al, Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability, In: 30th ICSE'08, pages 261–270, Leipzig, Germany, 2008.

[12] Hendrickson, S. and Hoek, A: Modeling Product Line Architectures through Change Sets and Relationships. Proc. ICSE 2007, pp. 189-198, Minneapollis, EUA, 2007.

[13] Nuseibeh, J. Kramer, and A. Finkelstein.Viewpoints: meaningful relationships are difficult. In ICSE '03: Proc of the 25th ICSE, pp. 676–681, Washington, USA, 2003.