# Towards a Generic and Extensible Merge Operator

**Andrew Jackson[1], Olivier Barais[2], Jean-Marc Jézéquel[2], Siobhán Clarke[1]**

[1]Distributed Systems Group, Dept. of Computer Science, Trinity College Dublin, Ireland.
{Andrew.Jackson, Siobhan.Clarke} @cs.tcd.ie
[2]IRISA, Campus Beaulieu, 35042 Rennes Cedex, France.
{barais, jezequel} @irisa.fr

## ABSTRACT

Merging is a common way to compose both crosscutting and non-crosscutting models. In this paper, we argue that merge can be defined more generically as an operator at the meta-modelling level. By describing merge at this level, a merge operator can be used to compose models based on meta-models other than UML. There are various merge variants and we concede that a full unification of all merge semantics may be infeasible. To define a common merge, we propose the definition of a common merge kernel as a semantic base that can be extended to realise the different expressions of merge.

## 1. INTRODUCTION

There have been many approaches to Aspect Oriented Modelling (AOM), each of which attempts to capture and address a significant issue or issues relating to crosscutting in design. We have surveyed twenty-two AOM approaches [1] and have found that all approaches to AOM are based on one conceptual common composition model - Merge.

In [2 p30-38], we demonstrated the commonality of merge through a detailed investigation of specific AOM approaches. From this investigation, we established that a merge composition specification enables both the behavioural and structural integration of both crosscutting and non-crosscutting design models.

We also noted that although a number of merge variants exist, the precise semantics of merge have typically not been expressed in literature. Theme/UML and PackageMerge are the primary sources of merge semantics for our work. There are significant differences between these merge semantics and we have previously described their unification to form an integrated semantic base of a common expression of merge [2].

The unification of these semantics is a proposal for merge based on elements defined in the UML meta-model. In this paper, our position is that merge can be designed more generically at the meta-model level. By describing merge at the meta-modelling level, a merge operator can be used to compose models based on various meta-models e.g., the UML meta-model and SDL meta-model.

There are, of course, diverse variants of merge and we concede that a full unification of all merge semantics may be infeasible. Consequently, we propose the design of a framework for the merge operator. This framework defines a common kernel merge as a semantic base, where extensions realise the different variants of merge semantics for different meta-models.

The remainder of this paper is organized as follows. Section 2 presents our initial work in which we define an architecture for implementing the merge operator in KerMeta [3]. Section 3 describes how the designer can use the framework to extend the semantics of the merge operators and enrich conflict detection for different meta-models. Finally, Section 4 concludes this short paper and presents our future work.

## 2. KERMETA - MERGE ARCHITECTURE

In this section, we present our initial work on implementing merge in KerMeta [3]. Firstly, we justify the choice of KerMeta to express the merge at the meta-level and next we present the architecture of the merge framework.

**KerMeta**: is meta-modelling language designed as an extension to the EMOF 2.0. KerMeta extends EMOF with an action language that supports the specification of semantics and behaviour of meta-models. The action language is imperative and object-oriented. It is used to provide an implementation of operations defined in meta-models. A more detailed description of the language is presented in [3].

The KerMeta action language has been specially architected to support the design of meta-models and meta-model behaviours. It includes both Object Oriented (OO) features and features specific to models. KerMeta includes traditional OO static typing, multiple inheritance and behaviour redefinition/selection with a late binding semantics. To make KerMeta suitable for model processing, more specific concepts such as opposite properties (i.e. associations) and handling of object containment have been included. In addition to this, convenient Object Constraint Language constructs, such as closures (e.g. each, collect, select), are available in KerMeta.

KerMeta has been chosen as an implementation platform for a generic merge operator for two reasons. Firstly, KerMeta supports the development of structural meta-models with behavioural semantics, and as such we can define both a structural and behavioural merge at the meta-model level. Secondly, KerMeta tools are compatible with the Eclipse Modelling Framework (EMF) that allows us to manage all the models that have a meta-model defined with EMF and to use Eclipse tools to edit, store, and visualize models.

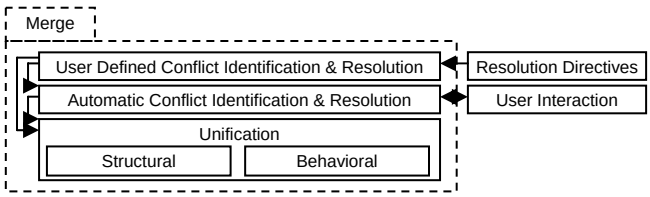


**Figure 1 Merge Architecture**

**Merge**: Figure 1 presents a layered architectural view of our merge framework. There are thee layers:

1. User Defined Conflict Identification & Resolution – this implements semantics for resolving conflicts identified in resolution directives that are defined by the user.

2. Automatic Conflict Identification & Resolution – this layer implements semantics for automatically identifying conflicts between models and either applying default resolution strategies or choosing an appropriate strategy to resolve the conflict.

3. Unification – This layer implements semantics for the integration of models. This layer can be split in to sub-layers: structural unification and behavioural unification. An example of structural unification is the integration of structures as defined by PackageMerge [2 p37]. An example for the behavioural unification is illustrated by Theme/UML [2, p31-34].

## 3. MERGE EXTENSIBILITY

**Overview:** The second layer of the framework is extensible in order to enrich the designer defined conflict resolution & identification layer. In the first case, the designer can define precisely the composition semantics by specifying how the conflicts are resolved. In the second case, he can configure the framework for different meta-models.

Figure 2 illustrates the behaviour of our merge framework. The merge framework is comprised of three main roles: the C*onflict Solver*, the C*onflict Finder*, and the *Conflict Fixers*. The *Solver* is a coordinator. It asks the *Finder* to detect any conflict and it delegates the resolution conflict to a *Fixer*. Fixers are related to specific types of conflict and encapsulate logic to resolve that specific conflict type.

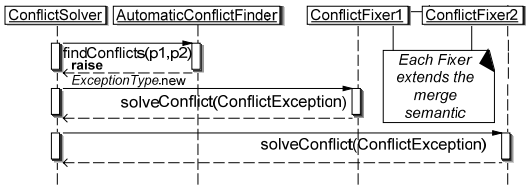


**Figure 2: sequence diagram of the conflict identification**

**Extensibility of the conflict identification & resolution layers:** All layers of the merge semantics can be easily extended in this framework. At its most basic (unification) level, the merge operator expects no conflicts and will fail to compose conflicting models (however this layer can be extended to compose new elements in the meta-model). By extending the automatic conflict resolution layer the designer can define default resolution strategies for specific types of conflicts. Such extensions are realised by implementing new *Fixers* and registering them with the *Conflict Solver*. In this implementation, it is necessary to inherit the abstract KerMeta class: *Fixer*. For example, designer can create a *Fixer* for the case where the *isAbstract* properties of classes of the same name are in conflict. To resolve the conflict, the *Fixer* will change the value of the *isAbstract* property to ensure conformity. Following the same pattern, the designer can create new *Fixers* for (re)solving other conflicts such as cases where methods that overlap one another conflict when merging two classes.

We propose two implementations of the package merge framework. The first implementation is based on the KerMeta (eMof) meta-model. Knowledge of the meta-model allows the designer to clearly identify all the possible conflicts and to create an exception type for each of these potential conflicts. The second implementation focuses on the model later and uses reflexivity to compare two models instances of the same meta-model. In this second case, it is more difficult (within the framework) to create exception types for each potential conflict. Through the framework, the designer can register a new exception type for each potential conflict. Typically, each potential conflict needs to be identified with the full qualified name of the property which is in conflict between the two models.

**Benefits:** Our merge framework provides two main advantages. Firstly, by implementing the merge framework as a layered architecture we ensure that:

1. The semantics defined at one layer in the architecture can be quickly (due to their isolation) and easily changed or extended (to ensure that new semantics or semantic variations can be easily catered for).

2. The designer can use only the layers that are relevant to their context. For example, the AO designer may not need conflict resolution facilities (this is noted in [2 p39]).

Moreover, the framework provides two dimensions of extensibility. First, the designer can extend the conflict resolution layer by creating new *Fixers.* He can also extend the conflict detection layer by implementing new *Exceptions*. Thanks to these two dimensions of extensibility, merge can be used to define a specific semantic of the eMof package Merge and can also be used to extend the conflict identification layer in order to address different meta-models.

## 4. SUMMARY & FUTURE WORK

There are diverse variants of merge and full unification of all merge semantics may be infeasible. To define a common merge, we have proposed the definition of a common kernel merge as a semantic base that can be extended to realize the different expressions of merge. The main advantage of this is the ability to compose models instances of different meta-models.

To evaluate our work, we used this framework to implement the semantics of the composition of Theme/UML in KerMeta. Next, we will demonstrate how our general merge can be applied to different meta-models (e.g. UML 2.0 & SDL). We also aim to use our framework to define the semantics of the UML2.0 package merge as well as other variations on merge.

## ACKNOWLEDGMENTS

This work is supported by European Commission grant IST-2-004349: European Network of Excellence on Aspect-Oriented Software Development (AOSD-Europe), 2004-2008

## 5. REFERENCES

[1] Jackson A., Clarke S., “Survey of AO analysis and design approaches”, , AOSD-Europe Deliverable D11, AOSD-Europe-ULANC-9, 18 May 2005, pp 154-236

[2] Jackson A., Clarke S., “Initial Version of Aspect-Oriented Design Approach”, , AOSD-Europe Deliverable D38, AOSD-Europe-TCD-7, 15 February 2006, pp 1-73

[3] Muller P.A., Fleurey F., Jézéquel J.M., “Weaving executability into object-oriented meta-languages”, in Proc of MODELS/UML'2005, volume 3713 of *LNCS*, pages 264-278, Montego Bay, Jamaica, October 2005. Springer.