Journal of Software Engineering Research and Development
a SpringerOpen Journal

RESEARCH Open Access

Model-based reuse for crosscutting frameworks: assessing reuse and maintenance effort

Thiago Gottardi[1*], Rafael Serapilha Durelli[2], Óscar Pastor López[3] and Valter Vieira de Camargo[1]

*Correspondence:
thiago_gottardi@dc.ufscar.br
[1]Departmento de Computação, Universidade Federal de São Carlos, Caixa Postal 676, 13.565-905, São Carlos, São Paulo, Brazil
Full list of author information is available at the end of the article

Abstract

Background: Over the last years, a number of researchers have investigated how to improve the reuse of crosscutting concerns. New possibilities have emerged with the advent of aspect-oriented programming, and many frameworks were designed considering the abstractions provided by this new paradigm. We call this type of framework Crosscutting Frameworks (CF), as it usually encapsulates a generic and abstract design of one crosscutting concern. However, most of the proposed CFs employ white-box strategies in their reuse process, requiring two mainly technical skills: (i) knowing syntax details of the programming language employed to build the framework and (ii) being aware of the architectural details of the CF and its internal nomenclature. Also, another problem is that the reuse process can only be initiated as soon as the development process reaches the implementation phase, preventing it from starting earlier.

Method: In order to solve these problems, we present in this paper a model-based approach for reusing CFs which shields application engineers from technical details, letting him/her concentrate on what the framework really needs from the application under development. To support our approach, two models are proposed: the Reuse Requirements Model (RRM) and the Reuse Model (RM). The former must be used to describe the framework structure and the later is in charge of supporting the reuse process. As soon as the application engineer has filled in the RM, the reuse code can be automatically generated.

Results: We also present here the result of two comparative experiments using two versions of a Persistence CF: the original one, whose reuse process is based on writing code, and the new one, which is model-based. The first experiment evaluated the productivity during the reuse process, and the second one evaluated the effort of maintaining applications developed with both CF versions. The results show the improvement of 97% in the productivity; however little difference was perceived regarding the effort for maintaining the required application.

Conclusion: By using the approach herein presented, it was possible to conclude the following: (i) it is possible to automate the instantiation of CFs, and (ii) the productivity of developers are improved as long as they use a model-based instantiation approach.

1 Content

This article is organized as follows: In Section 2 is presented the introduction of this article. Section 3 presents the necessary background to understand this article. More specifically, it is split into three sections, they are: Section 3.1 presents the concepts of Model-Driven Development, Section 3.2 showns the general notion of Aspect oriented programming and in Section 3.3 is presented the concepts of Crosscutting frameworks. In Section 4 is presented the proposed approach. In Section 5 is presented the evaluation of our approach. In Section 7 is presented some related works. Finally, in Section 8 we present the conclusion of this article.

2 Introduction

Aspect-Oriented Programming (AOP) is a programming paradigm that overcomes the limitations of Object- Orientation (Programming) providing more suitable abstractions for modularizing crosscutting concerns (CC) such as persistence, security, and distribution. AspectJ is one of the programming languages that implements these abstractions (AspectJ Team 2003). Since the advent of AOP in 1997, a substantial effort has been invested in discovering how such abstractions can enhance reuse methodologies such as frameworks (Fayad and Schmidt 1997) and product lines (Clements and Northrop 2002). One example is the research that aims to design a CC in a generic way so that it can be reused in other applications (Bynens et al. 2010; Camargo and Masiero 2005; Cunha et al. 2006; Huang et al. 2004; Kulesza et al. 2006; Mortensen and Ghosh 2006; Sakenou et al. 2006; Shah and Hill 2004; Soares et al. 2006; Soudarajan and Khatchadourian 2009; Zanon et al. 2010). Because of the absence of a representative taxonomy for this kind of design, in our previous work we have proposed the term "Crosscutting Framework" (CF) to represent a generic and abstract design and implementation of a single crosscutting concern (Camargo and Masiero 2005).

Most of the CFs which are found in the literature adopt white-box reuse strategies in their reuse process, relying on writing source code to reuse the framework (Bynens et al. 2010; Camargo and Masiero 2005; Cunha et al. 2006; Huang et al. 2004; Kulesza et al. 2006; Mortensen and Ghosh 2006; Sakenou et al. 2006; Shah and Hill 2004; Soares et al. 2006; Soudarajan and Khatchadourian 2009; Zanon et al. 2010). This strategy is flexible in terms of framework evolution; however, application engineers need to cope with details not directly related to the requirements of the application under development. Therefore, the following problems exist when using such strategies: (i) the learning curve is steep because application engineers need to learn the programming paradigm employed in the framework design; (ii) a number of errors can be inserted because of the manual creation of the source code.; (iii) the development productivity is negatively affected as several lines of code must be written to define a small number of hooks, and (iv) the reuse processes can only be initiated during the implementation phase as there is no source code available in earlier phases.

To overcome these problems, we present a new approach for supporting the reuse of CFs using a Model-Driven Development (MDD) strategy. MDD consists of a combination of generative programming, domain-specific languages and model transformations. MDD aims at reducing the semantic gap between the program domain and its implementation, using high-level models that screen software developers from complexities of the underlying implementation platform (France and Rumpe 2007). Our approach is based on two models: the Reuse Requirements Model (RRM) and the Reuse Model (RM). Built by a framework engineer, RRM documents all the features and variabilities of a CF. Application engineers can then select just the desired features from the RRM and generate a more specific model, referred to as the RM. Later, the application engineer can conduct the reuse process by completing the RM fields with information from the application and automatically generate the reuse code.

Furthermore, we present the results of two comparative experiments which used the same Persistence CF (Camargo and Masiero 2005). The first experiment aimed to compare the productivity of conducting a reuse process when using our model-based approach versus the ad-hoc approach, i.e., writing the source code manually. The purpose of the second experiment was to compare the effort of maintaining applications developed with both our model-based approach versus the ad-hoc way. Our approach presented clear benefits for the instantiation time (productivity); however, no differences were identified regarding the maintenance effort. Therefore, the main contribution of this paper is twofold: (*i*) introduction of a model-based approach for supporting application engineers during the reuse process of CFs and (*ii*) presentation of the results of two experiments.

3 Background

This section describes the background necessary to understand our proposed models. It is split into three subsections: the first one contains the concepts of Model-Driven Development, the second subsection has a basic description of aspect-oriented programming and the third one exposes the general notion of Crosscutting Frameworks.

3.1 Model-driven development

Software systems are becoming increasingly complex as customers demand richer functionality be delivered in shorter timescales (Clark et al. 2004). In this context, Model-Driven Development (MDD) can be used to speed up the software development and to manage its complexity in a better way by shifting the focus from the programming level to the solution-space.

MDD is an approach for software development that puts a particular emphasis upon making models the primary development artifacts and upon subjecting such models to a refinement process by using automatic transformations until a running system is obtained. Therefore, MDD aims to provide a higher abstraction level in the system development which further results in the improved understanding of complex systems (Pastor and Molina 2007).

Furthermore, MDD can be employed to handle software development problems that originate from the existence of heterogeneous platforms. This can be achieved by keeping different levels of model abstractions and by transforming models from Platform Independent Models (PIMs) to Platform Specific Models (PSMs) (Pastor and Molina 2007). Therefore, the automatic generation of application specific code offers many advantages such as: a rapid development of high quality code; a reduced number of accidental programming errors and the enhanced consistency between the design and the code (Schmidt 2006).

It is worth highlighting that models in MDD are usually represented by a domain-specific language (Fowler 2010), i.e., a language that adequately represents the information of a given domain. Instead of representing elements using a general purpose language

(GPL), the knowledge is described in the language which domain experts understand. Besides, as the experts use a suitable language to describe the system at hand, the accidental complexity that one would insert into the system to describe a given domain is reduced, leaving just the essential complexity of the problem.

3.2 Aspect-Oriented Programming

Aspect-Oriented Programming (AOP) aims at improving the modularization of a system by providing language abstractions that are dedicated to modularize crosscutting concerns (CCs). CCs are concerns which cannot be accurately modularized by using conventional paradigms (Kiczales et al. 1997). Without proper language abstractions, crosscutting concerns become scattered and tangled with other concerns of the software, affecting maintainability and reusability. In AOP, there is usually a distinction between base concerns and crosscutting concerns. The base concerns (or Core-concerns) are those which the system was originally designed to deal with. The crosscutting concerns are the concerns which affect on other concerns. Examples of crosscutting concerns include global restrictions, data persistence, authentication, access control, concurrency and cryptography (Kiczales et al. 1997).

Aspect-Oriented Programming languages allow programmers to design and implement crosscutting concern decoupled from the base concerns. The AOP compiler has the ability to weave the decoupled concerns together in order to attain a correct software system. Therefore, on the source-code level, there is a complete separation of concerns and the final release delivers the functionality expected by the users.

In this work we have employed the AspectJ language (Kiczales et al. 2001), which is an aspect-oriented extension for Java, allowing the Java code to be compiled seamlessly by the AspectJ compiler. The main constructs in this language are: aspect - a structure to represent a crosscutting concern; pointcut - a rule used to capture join points of other concerns; advices - types of behavior to be executed when a join point is captured; and intertype declarations - the ability to add static declarations from the outside of the affected code. In our work, intertype declarations are used to insert more interface realizations into classes of the base concern.

3.3 Crosscutting frameworks

Crosscutting Frameworks (CF) are aspect-oriented frameworks which encapsulate the generic behavior of a single crosscutting concern (Camargo and Masiero 2005; Cunha et al. 2006; Sakenou et al. 2006; Soudarajan and Khatchadourian 2009). It is possible to find CFs to support the implementation of persistence (Camargo and Masiero 2005; Soares et al. 2006), security (Shah and Hill 2004), cryptography (Huang et al. 2004), distribution (Soares et al. 2006) and other concerns (Mortensen and Ghosh 2006). The main objective of CFs is to make the reuse of crosscutting concerns a reality and a more productive task during the development of an application.

As well as other types of frameworks, CFs also need specific pieces of information regarding the base application to be reused correctly and to work properly. We name this kind of information "Reuse Requirements" (RR). For instance, the RR for an Access Control CF includes: 1) the application methods that need to have their access controlled; 2) the roles played by users; 3) the number of times a user is allowed get an incorrect password. This information is commonly documented in manuals known as "Cookbooks".

Unlike application frameworks, which are used to generate a whole new application, a CF needs to be coupled to a base application to become operational. The conventional process to reuse a CF is composed by two activities: instantiation and composition. During the instantiation, an application engineer chooses variabilities and implements hooks, while during the composition, he/she provides composition rules to couple the chosen variabilities to a base code.

CF-based applications, i.e, applications which were developed with the support of CFs, are composed by three types of modules: a base code module, a reuse code module and framework itself. The "base code" represents the source code of the base application and the "framework code" is the CF source code, which is untouched during the reuse process. The "reuse module" is the connection between the base application and the framework and it is developed/written by the application engineer. Applications can be composed by several CFs, each one coupled by one reuse module. The source code created specifically to reuse a CF, is referred here as "reuse code".

In our previous work we have developed a Persistence CF (Camargo et al. 2004) which is used here as a case study. This CF was designed like a product-line, so it has certain mandatory features, for instance, "Persistence" and "Connection". The first one aims to introduce a set of persistence operations (e.g., store, remove, update, etc) into application persistence classes. The second feature is related to the database connection and identifies points in the application code where a connection needs to be established or closed. This feature has variabilities as the Database Management System (e.g., MySQL, SyBase, Native and Interbase). This CF also has a set of optional features such as "Caching", which is used to improve the performance by keeping copies of data in the local memory, and "Pooling", which represents a number of active database connections.

4 Model-based reuse approach

In this section we present our approach and the models that support during the instantiation and composition of CFs: Reuse Requirements Model (RRM) and Reuse Model (RM). These models have been formulated on top of Eclipse Modeling Framework and Graphical Modeling Framework (Eclipse Consortium 2011). The formal definition of both models is specified by the metamodel shown in Figure 1. It is comprised of a set of enumerations, abstract and concrete metaclasses.

The metamodel was built based on the vocabulary commonly used in the context of CFs, for example: pointcuts, classifier extensions, method overriding, and variability selection. These concepts were mapped into concrete metaclasses, which are visible under the dashed line of Figure 1.

Above the dashed line, there are also the following enumerations: "Visibility", "SuperType" and "CompositionType", which are sets of literals used as metaclass properties. The other elements above the line are abstract metaclasses, which were created after generalizing the properties of the concrete metaclasses. These abstract metaclasses can be applied in similar approaches and are also important to improve modularity and to avoid code replication of the reuse code generator.

Both of our proposed models are identical, however they are employed in different moments of the process. The first proposed model, the RRM, is a graphical documentation for Reuse Requirements, i.e., it graphically documents all the information needed to couple a CF to a base application. Conventionally, this is known as "cookbooks". This

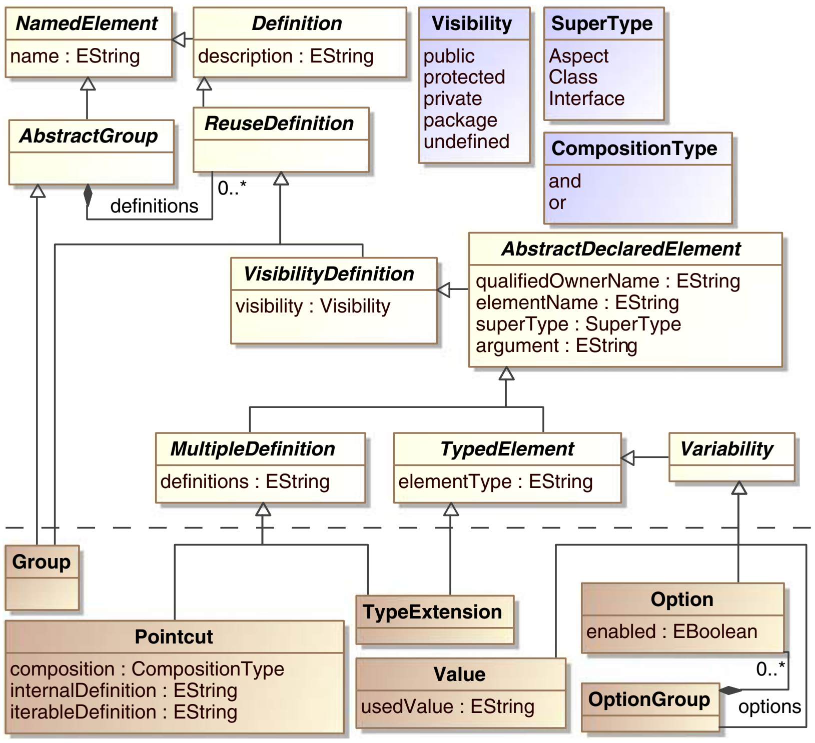
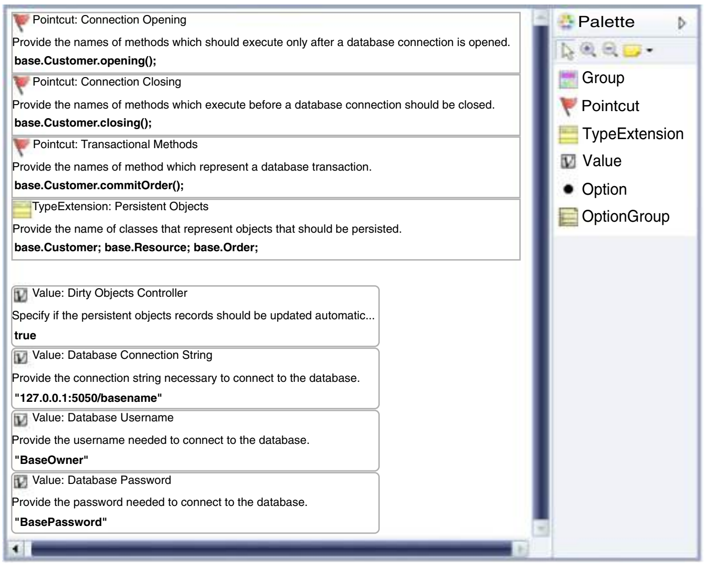


Figure 1 Metamodel of the proposed models. A metaclass diagram of 'CFReuse', which contains all metaclasses and enumerations.

model involves information regarding all CF features and must be developed/provided by a framework engineer. The second model, the RM, is a subset of the RRM and contains only the selected features for conducting a reuse process. Since both models share the same metamodel, it is possible to employ a direct model transformation to instantiate a RM from a RRM by selecting a valid set of features. Both of our models are represented as forms containing boxes, as seen in Figure 2. Each box is an instance of a concrete meta-class element and represents a reuse requirement. Each box contains three lines. The first one contains both an icon representing the type of the element, (which is the same type visible in the "Palette") and the name of the reuse requirement. The second line shows a description and the last line must be filled by the application engineer to provide the necessary information regarding the base application. Notice that the last line is used only in RMs.

By analyzing a RRM, the application engineer can identify all the information required by the framework to conduct the reuse process. For example, this model represents the variabilities that must be chosen by the application engineer and also indicates join-points of the base code where crosscutting behavior must be applied to, as well as classes, interfaces, or aspect names that must be affected.

Framework variabilities that must be chosen during reuse process are also visible. For example, to instantiate a persistence CF, several activities must be done, among them: i) informing points of the base application in which the connection must be open and closed; ii) informing methods that represent data base transactions and iii) choosing variabilities, e.g., the driver that should be used to connect to the database.


Pointcut: Connection Opening
Provide the names of methods which should execute only after a database connection is opened.
base.Customer.opening();

Pointcut: Connection Closing
Provide the names of methods which execute before a database connection should be closed.
base.Customer.closing();

Pointcut: Transactional Methods
Provide the names of method which represent a database transaction.
base.Customer.commitOrder();

TypeExtension: Persistent Objects
Provide the name of classes that represent objects that should be persisted.
base.Customer; base.Resource; base.Order;

Value: Dirty Objects Controller
Specify if the persistent objects records should be updated automatic...
true

Value: Database Connection String
Provide the connection string necessary to connect to the database.
"127.0.0.1:5050/basename"

Value: Database Username
Provide the username needed to connect to the database.
"BaseOwner"

Value: Database Password
Provide the password needed to connect to the database.
"BasePassword"

Figure 2 Reuse requirements models and a reuse models editor. A graphical user interface of the model editor. The editor is capable of editing Reuse Requirement Models and Reuse Models. The pallete on the right of the figure lists every concrete metaclass, which can be modelled on model level by dragging and dropping.

The another model, the RM, is shown in Figure 2. It supports the reuse process of a crosscutting framework by filling in the third line of the boxes. Therefore, RM must be used by the application engineer to reuse a framework. For instance, the value "*base.Customer.opening()*" is a method of the base application that was inserted by the application engineer into the third line of the "Connection Opening" box to inform that the DB connection must be established before this method runs.

The code generator transforms the Reuse Model into the Reuse Code, which consists of pieces of AspectJ code used to couple the base application to the crosscutting framework. This transformation is not a one to one conversion, i.e., every element in the model not always generates the same number of code elements. This was a special underlying challenge we have experienced when implementing this approach. The code generator needs to read the RM completely and to aggregate all data to identify how many files need to be generated.

The reuse model elements contain attributes to define the super classes to be extended; several elements may identify the same superclass. Therefore, the code generator must identify every superclass in order to create a single subclass per superclass when generating "Pointcuts", "Options" and "Value Definitions".

The generation of "Type Extensions" is slightly different. Whenever there is a single type extension, the code generator creates a single aspect that aggregates every type extension using "declare parents"; a specific type of intertype declaration.

The architecture of the generator is represented in Figure 3. Initially, the XTend (Efftinge 2006) library is used as a front end of the compiler, loading the data of the model into a hierarchical structure in memory, similar to a Domain Object Model. After the structure is loaded, it is processed in order to identify the units that must be generated. This process creates another structure that represents the resulting code, which is similar to an abstract syntax tree. The "AJGenerator" is a back end of the generator that we have also created; it is capable of transforming this tree into actual files of valid AspectJ code.

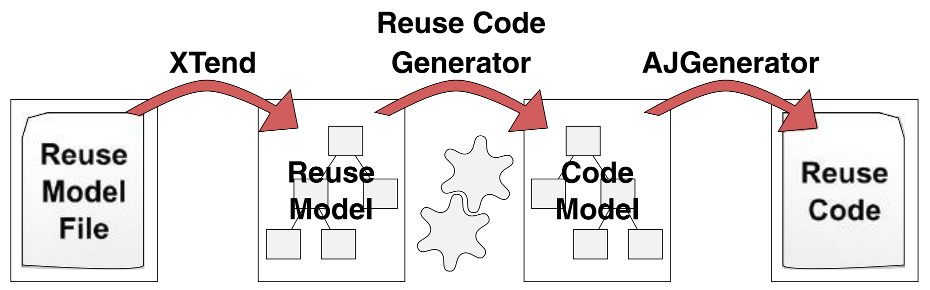


Figure 3 Reuse code generator architecture. The reuse code generator creates a reuse code by processing a reuse model. This process is divided into three sub processes, which are shown in this figure.

4.1 Reuse process

This subsection explains the reuse process that is defined when using the new proposed models (RRM and RM). From this point it is important to clarify the distinction between the terms *model* and *diagram*. Model is a more generic term and it is physically represented by XML files, while a diagram is a visual representation of a model. So, in our case, the Reuse Requirements Diagram (RRD) is a diagram that represents the Reuse Requirements Model and the Reuse Diagram (RD) is a diagram that represents the Reuse Model. It is also worth mentioning that these diagrams are similar to forms, in which they must be filled in. In order to explain the new process, there is an activity diagram in Figure 4 illustrating the perspective of both developers: framework engineers and application engineers.

Since the CF must be completely defined before its reuse process is started, this explanation begins from the framework engineer's point of view. At the right side of the Figure 4, the framework engineer starts developing a new CF for a specific crosscutting concern. The first activity is to develop the framework itself (marked with 'A'). Then, the engineer should make the CF code available for reuse ('B') and should create the RRD ('C'), graphically indicating the information required to couple his CF to a base application. This diagram ('D') will be available for the application engineer. Upon finishing this process, the framework engineer has two artifacts that will be used by the application engineer: the Reuse Requirements Diagram ('D') and the Framework code ('B').

The reuse process starts on the left side of the figure, where the perspective of application engineers is considered. This engineer is responsible for developing the application, which is composed by both the "Base" and "Reuse" modules. By analyzing the application being developed ('a'), the application engineer must identify the concerns that would affect the software, possibly by using an analysis diagram ('b'). By having these concerns identified, the application engineer is able to select the necessary frameworks and to start the reuse process since the earlier development phases. After selecting and analyzing the RRD of the selected frameworks ('c'), it is necessary to select a subset of the optional variabilities ('d') because some elements may not be necessary (since the framework may be supplied with default values), or to select mutually exclusive features. The selected elements will be carried to a new "Reuse" diagram ('e'). If there are more than one CF being reused, then there should be a "Reuse" diagram for each one of them. The application engineer should then design the base application ('g') documenting the name of the units,

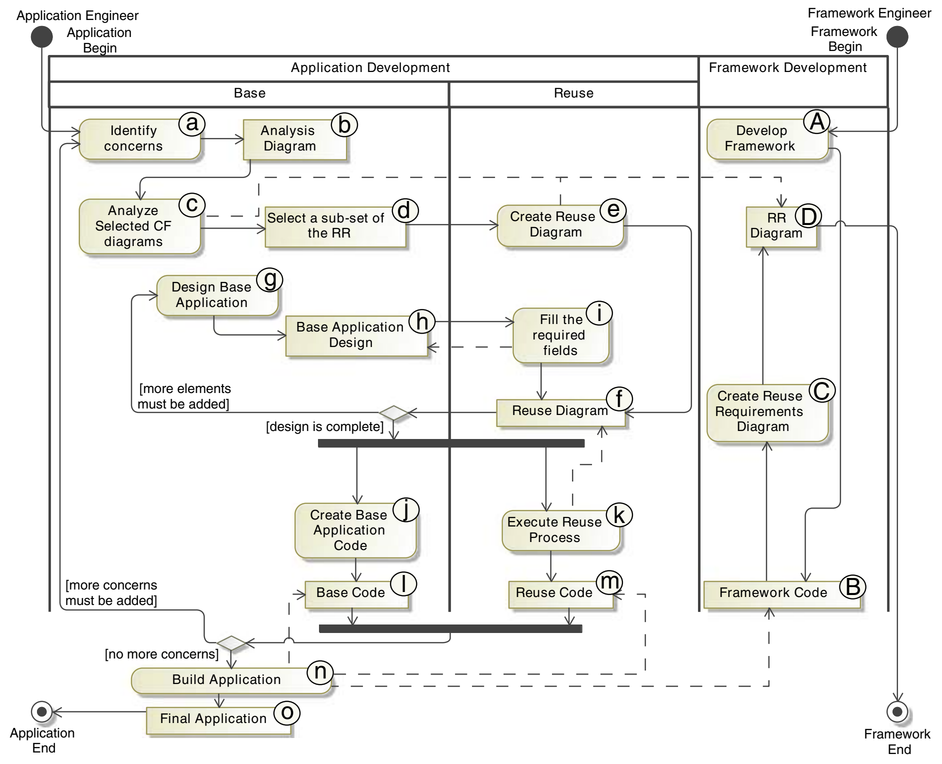


Figure 4 Reuse process activity diagram. A UML activity diagram that represents the reuse process to be employed when using the proposed models and the tools.

methods and attributes found on the base application (‘h’). By designing the names of elements needed by the framework, they will become available, meaning that it is already possible to enter these names in the RD. This should be done before all required elements of the iteration are designed. After defining these names, which are the values needed by the reuse portion, they must be filled (‘i’) in the reuse diagram (‘f’) to enable the coupling among the modules.

The base application can be developed (‘j’) in parallel with the reuse process execution (‘k’), which is a model transformation to generate the “Reuse Code” (‘m’) from the “Reuse Diagram” (‘f’). After completing the “Base Code” (‘l’) and the “Reuse Code” (‘m’), the application engineer may choose between adding a new concern (and extending the base application) or finishing the process. At that moment, the following pieces of code are available: the “Base Code” (‘l’), the “Reuse Codes” (‘m’) and the selected “Framework Codes” (‘B’). All of these codes are processed to build (‘n’) the “Final Application” (‘o’) and to conclude the process.

The transformation employed to create the RD avoids manual creation of this model. This is possible by identifying the selected framework and by processing its RRD. Besides accelerating the creation of this model, this also allows the RD to take all the needed elements from the earlier diagram to the code generation. However, the values regarding the base application are still needed and must be informed by the application engineer. The RRD contains information needed by the framework being reused. By identifying that information during earlier development phases it is easier to define it correctly. Consequently, the base application is not oblivious of the framework and its behaviors, however, the modules are completely isolated and have no code dependency among them. It is important to point out that the Reuse Code itself depends on the Base Code during the creation process, however, its definition can be made as soon as the base application design is complete.

4.2 Approach usage example

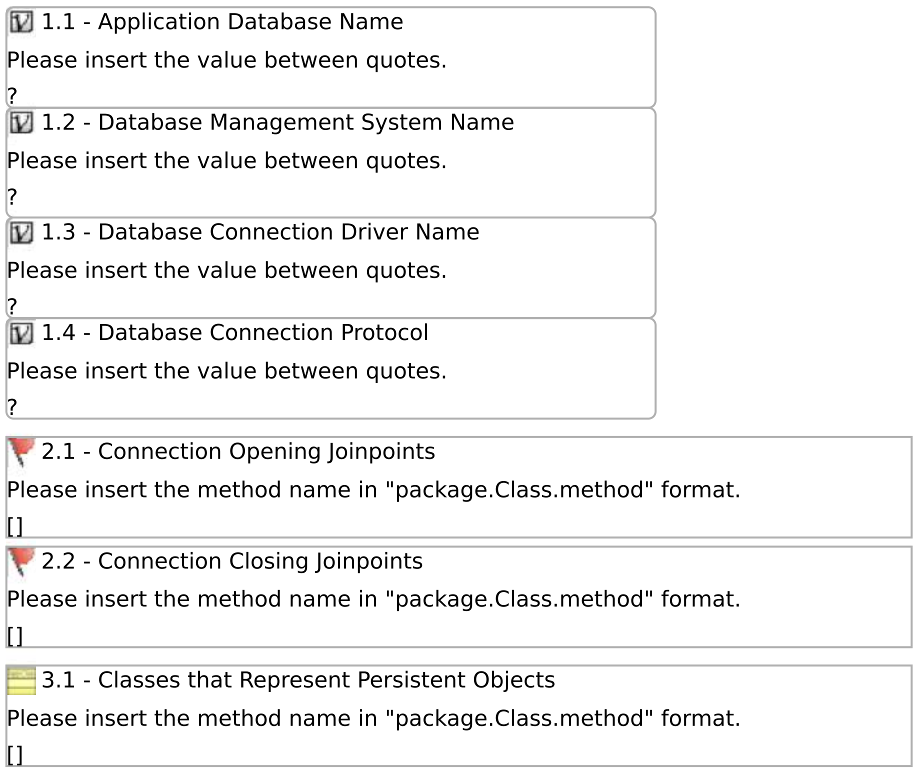
An usage example of our approach is described in this section. Firstly, we briefly describe the domain engineering which contains the creation of the framework reuse model. Finally, the application engineering is described, which consists of reuse model completion and reuse code generation, thus completing the process.

4.3 Domain engineering

The domain engineer must create a reuse model which contains the information necessary to reuse a crosscutting framework. In the example provided herein, every information needed to create a reuse model for a persistence framework. After the model creation, its completion is shown during application engineering to reuse the framework and couple it to an example application.

The reuse model template for the crosscutting framework in Figure 5, which was derived from a reuse requirements model by describing the framework hotspots. In Figure 6, the reuse model is shown after its completion.

The model elements are defined as follows: there are four value objects, two pointcut objects, and one type extension object. The value objects are used to define strings needed



1.1 - Application Database Name
Please insert the value between quotes.
?

1.2 - Database Management System Name
Please insert the value between quotes.
?

1.3 - Database Connection Driver Name
Please insert the value between quotes.
?

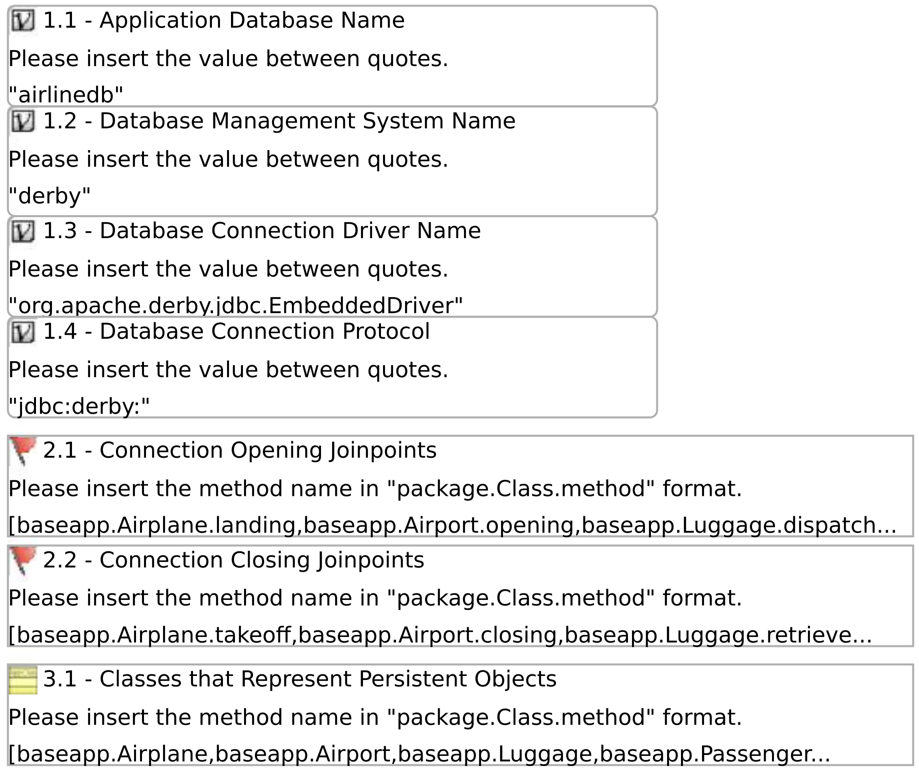
1.4 - Database Connection Protocol
Please insert the value between quotes.
?

2.1 - Connection Opening Joinpoints
Please insert the method name in "package.Class.method" format.
[]

2.2 - Connection Closing Joinpoints
Please insert the method name in "package.Class.method" format.
[]

3.1 - Classes that Represent Persistent Objects
Please insert the method name in "package.Class.method" format.
[]

Figure 5 Reuse model template. A reuse model template that must be provided by the domain engineer in order to allow the application engineers to apply the process.



1.1 - Application Database Name
Please insert the value between quotes.
"airlinedb"

1.2 - Database Management System Name
Please insert the value between quotes.
"derby"

1.3 - Database Connection Driver Name
Please insert the value between quotes.
"org.apache.derby.jdbc.EmbeddedDriver"

1.4 - Database Connection Protocol
Please insert the value between quotes.
"jdbc:derby:"

2.1 - Connection Opening Joinpoints
Please insert the method name in "package.Class.method" format.
[baseapp.Airplane.landing,baseapp.Airport.opening,baseapp.Luggage.dispatch...

2.2 - Connection Closing Joinpoints
Please insert the method name in "package.Class.method" format.
[baseapp.Airplane.takeoff,baseapp.Airport.closing,baseapp.Luggage.retrieve...

3.1 - Classes that Represent Persistent Objects
Please insert the method name in "package.Class.method" format.
[baseapp.Airplane,baseapp.Airport,baseapp.Luggage,baseapp.Passenger...

Figure 6 Complete reuse model. A complete reuse model after the application engineer provides the information regarding the base application, which will allow framework reuse by code generation.

by the framework in order to connect it to the database. They are used to define the database name, the name of the database management system, the database connection driver, and the database connection protocol. Every property of these items are then represented on Tables 1, 2, 3 and 4.

The pointcut objects are used to define joinpoints of the base application. The first pointcut object is represented on Table 5 and it must be used to inform where DB connections must be established. To do that, the application engineer needs to inform which methods execute right after a DB connection is established, i.e., methods that operate properly only if there is a connection open. The second point cut object is represented on Table 6 and it is used to inform methods that execute right before the connection is closed, therefore, the last method that needs an open connection.

The last object is represented on Table 7, which is used to define the classes found in the base application. These base application define object types that must be persisted on the database.

This reuse model is provided along with the crosscutting framework to be used by the application engineer in order to instantiate the framework, which is described in Section 4.3.

4.4 Application engineering

An example of an application development is given in this subsection. This application is referred to as Airline Ticket Management and must be coupled to the persistence CF

Table 1 Application database name

Value object: application database name	
Name	***Name***
"1.1 - Application database name"	
Description	***Description***
"Please insert the value between quotes."	
Qualified name of the owner	***QualifiedOwnerName***
"persistence.instantiation.helper.ExtendedConnectionVariabilities"	
Method to be overridden	***ElementName***
"ExtendedConnectionVariabilities.setDatabaseName"	
Value data type	***ElementType***
"String"	
Select supertype (aspect, class or interface)	***SuperType***
"Class"	

A string needed by the database connection API in order to connect to a specific database managed by the database system.

previously mentioned. This application uses the Apache Derby Database Management System (Apache Software Foundation 2012). The design of this application is shown on Table 8.

Upon the reuse model completion, the resulting reuse model is similar to that shown in Figure 7. Despite not being shown in the application details, every base application class was created inside the package "baseapp". After validating the model, the reuse code is generated; it is divided into three units.

The first generated unit is an aspect that extends a framework class. The overridden methods are used to return constant values that are necessary for the framework to successfully get connected to the database. It is important to emphasize that the four values have been defined in the same unit because they are owned by the same superclass. This would not happen if their superclasses were different.

The second unit is shown in Figure 8, which is an aspect that overrides pointcuts *openConnection* and *closeConnection*. These pointcuts are used to capture base

Table 2 Database management system name

Value object: database management system name	
Name	***Name***
"1.2 - Database management system name"	
Description	***Description***
"Please insert the value between quotes."	
Qualified name of the owner	***QualifiedOwnerName***
"persistence.instantiation.helper.ExtendedConnectionVariabilities"	
Method to be overridden	***ElementName***
"ExtendedConnectionVariabilities.setSpecificDatabase"	
Value data type	***ElementType***
"String"	
Select supertype (aspect, class or interface)	***SuperType***
"Class"	

A string needed by the database connection API in order to select the database connection driver.

Table 3 Connection driver protocol

Value object: database connection driver name	
Name	***Name***
“1.3 - Database connection driver name”	
Description	***Description***
“Please insert the value between quotes.”	
Qualified name of the owner	***QualifiedOwnerName***
“persistence.instantiation.helper.ExtendedConnectionVariabilities”	
Method to be overridden	***ElementName***
“ExtendedConnectionVariabilities.getDriver”	
Value data type	***ElementType***
“String”	
Select supertype (aspect, class or interface)	***SuperType***
“Class”	

A string needed by the database connection API in order to select the database connection driver.

application joinpoints that trigger the database connections and disconnections. They are defined in a single aspect because they also share the same superclass.

Figure 9 shows another aspect, which uses static crosscutting features to define classes that extend the interface specified by domain engineer by using the “Declare Parents” syntax.

Our model generator is also capable of generating a validation code, which checks if the base element names inserted into the reuse model are valid.

5 Methods

Two experiments have been conducted to compare our model-based reuse approach with the conventional way of reusing CFs, i.e., manually creating the reuse code. The first experiment is called Reuse Study and was planned to identify the gains in productivity when reusing a framework. The second experiment is denominated “Maintenance Study” and was designed to identify whether the our models help or not in the maintenance of

Table 4 Connection driver protocol

Value object: database connection protocol	
Name	***Name***
“1.4 - Database connection protocol name”	
Description	***Description***
“Please insert the value between quotes.”	
Qualified name of the owner	***QualifiedOwnerName***
“persistence.instantiation.helper.ExtendedConnectionVariabilities”	
Method to be overridden	***ElementName***
“ExtendedConnectionVariabilities.getJDBC”	
Value data type	***ElementType***
“String”	
Select supertype (aspect, class or interface)	***SuperType***
“Class”	

A string needed by the database connection API in order to connect to the database system.

Table 5 Connection opening joinpoint definition

Pointcut Object: connection opening joinpoints	
Name	***Name***
"2.1 - Connection opening joinpoints"	
Description	***Description***
"Please insert the method name in "package.Class.method" format."	
Qualified name of the owner	***QualifiedOwnerName***
"persistence.instantiation.helper.ExtendedConnectionCompositionRules"	
Pointcut to be overridden	***ElementName***
"openConnection"	
Composition operator	***Composition***
"or"	
Internal pointcut definition	***InternalDefinition***
""	
Iterable pointcut definition	***IterableDefinition***
"execution (* %s(..))"	
Select supertype (aspect, class or interface)	***SuperType***
"Class"	

The second pointcut object is used to define methods that run right after the database connection must be open.

a CF-based application. This second study is important because maintenance activities are usually performed more often than the reuse process. Each experiment has been performed twice. In this paper, the first execution is referred to as "First" and the second execution is referred to as "Replication". Since there have been only two executions for each experiment, we present four study executions in this section. The structure of the studies has been defined according to the recommendations of Wohlin et al. (2000).

Table 6 Connection closing joinpoint definition

Pointcut object: connection closing joinpoints	
Name	***Name***
"2.2 - Connection closing joinpoints"	
Description	***Description***
"Please insert the method name in "package.Class.method" format."	
Qualified name of the owner	***QualifiedOwnerName***
"persistence.instantiation.helper.ExtendedConnectionCompositionRules"	
Pointcut to be overridden	***ElementName***
"closeConnection"	
Composition operator	***Composition***
"or"	
Internal pointcut definition	***InternalDefinition***
""	
Iterable pointcut definition	***IterableDefinition***
"execution (* %s(..))"	
Select supertype (aspect, class or interface)	***SuperType***
"Class"	

The second pointcut object is used to define methods that run right before the database connection must be closed.

Table 7 Persistent objects definition

Type extension object: persistent objects	
Name	***Name***
"3.1 - Classes that represent persistent objects"	
Description	***Description***
"Please insert the class name in "package.Class" format."	
Qualified name of the owner	***QualifiedOwnerName***
"persistence.PersistentRoot"	
Select supertype (aspect, class or interface)	***SuperType***
"Interface"	

The last object from the example that is used to define the persistent objects of the application.

5.1 Reuse study definition

The objective was to compare the effort of reusing frameworks by using a conventional technique with the effort of using a model-based technique. The Persistence CF, briefly presented in Subsection 3.3, has played the role of "study subject" and it was used in both reuse techniques (conventional and model-based). The quantitative focus was determined

Table 8 Base application details

Constant definition		
Application database name	"airlinedb"	
Database management system name	"derby"	
Database connection driver name	"org.apache.derby.jdbc.EmbeddedDriver"	
Database connection protocol	"jdbc:derby:"	
Joinpoint definition (method execution)		
Joinpoints	**Method execution**	
Connection opening joinpoints	**Class**	**Method**
	Airplane	Landing
	Airport	Opening
	Luggage	Dispatch
	Checkin	Confirm
	Passenger	Onboard
	Flight	Depart
Connection closing joinpoints	**Class**	**Method**
	Airplane	Takeoff
	Airport	Closing
	Luggage	Retrieve
	Checkin	Cancel
	Passenger	Unboard
	Flight	Arrive
Type extensions		
Classes that represent persistent objects	**Class**	
	Airport	
	Luggage	
	Checkin	
	Passenger	
	Flight	

Details of a base application that needs to be coupled to the crosscuting framework.

```
                  ConcreteExtendedConnectionVariabilities0.aj

package persistence.reuse;
import persistence.instantiation.helper.ExtendedConnectionVariabilities;
public aspect ConcreteExtendedConnectionVariabilities0 extends
 ExtendedConnectionVariabilities {
    public String ExtendedConnectionVariabilities.setDatabaseName (){
        return "airlinedb";
    }
    public String ExtendedConnectionVariabilities.setSpecificDatabase (){
        return "derby";
    }
    public String ExtendedConnectionVariabilities.getDriver (){
        return "org.apache.derby.jdbc.EmbeddedDriver";
    }
    public String ExtendedConnectionVariabilities.getJDBC (){
        return "jdbc:derby:";
    }
}
```

Figure 7 Reuse code - first unit. The first generated reuse code fragment contains an aspect that is used to define the "Connection Variabilities" of the framework: these "Connection Variabilities" are provided by overriding methods.

considering the time spent in conducting the reuse process. The qualitative focus was to determine which technique takes less effort during the reuse process. This experiment was conducted from the perspective of application engineers reusing CFs: the study object was the 'effort' to perform a CF reuse.

5.2 Reuse study planning

The first experiment was planned considering the following question: "Which reuse technique takes less effort to reuse a CF?";

```
                ConcreteExtendedConnectionCompositionRules1.aj

package persistence.reuse;
import persistence.instantiation.helper.ExtendedConnectionCompositionRules;
public aspect ConcreteExtendedConnectionCompositionRules1 extends
    ExtendedConnectionCompositionRules {
    public pointcut openConnection ():
        (
                execution (* baseapp.Airplane.landing(..)) ||
                execution (* baseapp.Airport.opening(..)) ||
                execution (* baseapp.Luggage.dispatch(..)) ||
                execution (* baseapp.Checkin.confirm(..)) ||
                execution (* baseapp.Passenger.board(..)) ||
                execution (* baseapp.Flight.depart(..))
        );
    public pointcut closeConnection ():
        (
                execution (* baseapp.Airplane.takeoff(..)) ||
                execution (* baseapp.Airport.closing(..)) ||
                execution (* baseapp.Luggage.retrieve(..)) ||
                execution (* baseapp.Checkin.cancel(..)) ||
                execution (* baseapp.Passenger.unboard(..)) ||
                execution (* baseapp.Flight.arrive(..))
        );
}
```

Figure 8 Reuse code - second unit. The second generated reuse code fragment contains an aspect that is used to define the "Composition Rules" of the framework. The "CompositionRules" are provided by overriding pointcuts.

```
                                    ParentsDeclaration.aj

package  persistence.reuse;
public aspect  ParentsDeclaration   extends  Object {
    declare parents  : baseapp.Airplane    implements
        persistence.PersistentRoot;
    declare parents  : baseapp.Airport     implements
        persistence.PersistentRoot;
    declare parents  : baseapp.Luggage     implements
        persistence.PersistentRoot;
    declare parents  : baseapp.Checkin     implements
        persistence.PersistentRoot;
    declare parents  : baseapp.Passenger   implements
        persistence.PersistentRoot;
    declare parents  : baseapp.Flight      implements
        persistence.PersistentRoot;
}
```

Figure 9 Reuse code - third unit. The third generated reuse code fragment contains an aspect that is used to define static crosscutting features needed during the framework reuse. By default, all static crosscutting declarations are merged into a single aspect.

5.2.1 Context selection

Both studies have been conducted by students of Computer Science. In this section, they are referred to as "participants". Sixteen participants took part in the experiments, eight of those were undergraduate students and the other eight were post-graduate students. Every participant had a prior AspectJ experience.

5.2.2 Formulation of hypotheses

Table 9 contains our formulated hypotheses for the reuse study, which are used to compare the productivity of our tool with the conventional process.

There are two variables shown on the table: "Tc_r" and "Tm_r". "Tc_r" represents the overall time necessary to reuse the framework using the conventional technique while "Tm_r" represents the overall time necessary to reuse the framework using the model-based technique. There are three hypotheses shown on the table: "$H0_r$", "Hp_r" and "Hn_r". "$H0_r$" represents the null hypothesis, which is true when both techniques are equivalent; then, the time spent using the conventional technique minus the time spent

Table 9 Hypotheses for the reuse study

$H0_r$	There is no difference between using our tool and using an ad-hoc reuse process in terms of productivity (time) to successfully couple a CF with an application. Then, the techniques are equivalent. $Tc_r - Tm_r \approx 0$
Hp_r	There is a positive difference between using our tool and using an ad-hoc reuse process in terms of productivity (time) to successfully couple a CF with an application. Then, the conventional technique takes more time than the model-based tool. $Tc_r - Tm_r > 0$
Hn_r	There is a negative difference between using our tool and using an ad-hoc reuse process in terms of productivity (time) to successfully couple a CF with an application. Then, the conventional technique takes less time than the model-based tool. $Tc_r - Tm_r < 0$

Considering the Reuse Study, there are three hypotheses for the outcome. In the first one, both are equivalent, while in two of them, a technique is faster.

using the model-based tool is approximately zero. “Hp$_r$” represents the first alternate hypothesis, which is true when the conventional technique takes longer than the model-based tool; then, the time spent to use the conventional technique minus the time of the model-based tool is positive. “Hn$_r$” represents the second alternate hypothesis, which is true when the conventional technique takes longer than the model-based tool; then, the time taken to use the conventional technique minus the time taken to use the model-based tool is negative. As these hypotheses consider different ranges of a single resulting real value, then, they are mutually exclusive and only one of them is true.

5.2.3 *Variable selection*

The dependent variable in this work is the “time spent to complete the process”. The independent variables are Base Application, Technique and Execution Types, which, are controlled and manipulated.

5.2.4 *Participant selection criteria*

The participants were selected through a non-probabilistic approach by convenience, i. e., the probability of all population elements belong to the same sample is unknown. We have invited every student from the computing department of Federal University of São Carlos that attended the AOP course, a total of 17 students. Every student had to be able to reuse the framework by editing code during the training. Because of that, one undergraduate student was rejected before the execution.

5.2.5 *Design of the study*

The participants were divided into two groups. Each group was composed by four graduate students and four undergraduate students. Each group was also balanced considering a characterization form and their results from the pilot study. Table 10 shows the planned phases.

Table 10 Study design

Phase	Group 1	Group 2
General training	Reuse and maintenance training	
	Repair shop	
1st Reuse	Conventional	Models
Pilot phase	Hotel application	
2nd Reuse	Models	Conventional
Pilot phase	Library application	
1st First	Conventional	Models
Reuse phase	Deliveries application	
2nd First	Models	Conventional
Reuse phase	Flights application	
1st Replication	Conventional	Models
Reuse phase	Medical clinic application	
2nd Replication	Models	Conventional
Reuse phase	Restaurant application	
1st First	Conventional	Models

The Study Design contains every phase from both studies. It contains the sequence of operations, technique and the conidered applications.

5.2.6 Instrumentation for the reuse study

Base applications were provided together with two documents. The first document was a manual regarding the current reuse technique, and the second document was a list of details, which described the classes, methods and values regarding the application to be coupled.

The provided applications had the same reuse complexity. The participants had to specify four values, twelve methods and six classes in order to reuse the framework and to couple it to each application. These applications were designed with exactly the same structure of six classes. Each class contained six methods plus a class with a main method which is used to run the test case.

Each phase row of the Table 10 is divided into sub-rows that contain the name of the application and the technique employed to reuse the framework. For instance, during the First Reuse Phase, the participants of the first group coupled the framework to the "Deliveries Application" by using the conventional technique. The participants of the second used the model-based tool to perform the same exercise.

5.3 Operation for reuse study

5.3.1 Preparation

During the maintenance study, the students had to fix a reuse artifact to complete the process. Every participant had to fix every application by using only one of the techniques in equal numbers.

5.3.2 Execution

The participants had to work with two applications; each group started with a different technique. The secondary executions were replications of the primary executions with two other applications They were created to avoid the risk of getting unbalanced results during the primary execution, since some data that we gathered during the pilot study were rendered invalid.

5.3.3 Data validation

The forms filled by the participants were confirmed with the preliminary data gathered during the pilot study. In order to provide a better controllability, the researchers also watched the notifications from the data collector to check if the participants had concluded the maintenance process and had gathered the necessary data.

5.3.4 Data collection

The recorded timings during the reuse processes with both techniques are listed on the Table 11. Each table has five columns. Each column is defined by a letter or a word: "G" stands for the group of the participant during the activity; "A" stands for the application being reused; "T" stands for the reuse technique which is either "C" for conventional or "M" for model-based tool; "P" column lists an identifying code of the participants (students), whereas, the least, eight values are allocated to graduate students and the rest are undergraduate students; "Time" column lists the time the participant spent to complete each phase. The raw data we have gathered during the reuse study is also available as Additional file 1.

Table 11 Reuse process timings

Primary execution					Secondary execution				
G	**A**	**T**	**P**	**Time**	**G**	**A**	**T**	**P**	**Time**
1	Flights	M	15	04:19.952015	2	Clinic	M	10	02:59.467569
1	Flights	M	13	04:58.604963	1	Restaurant	M	13	03:56.785359
1	Flights	M	8	05:18.346829	1	Restaurant	M	15	04:23.629206
2	Delivery	M	11	05:24.249952	2	Clinic	M	11	04:25.196135
2	Delivery	M	5	05:31.653952	1	Restaurant	M	8	04:33.954349
2	Delivery	M	9	05:45.484577	2	Clinic	M	9	04:41.254920
2	Delivery	M	3	06:16.392424	1	Restaurant	M	12	05:05.524264
2	Delivery	M	10	06:45.968790	2	Clinic	M	3	05:45.333167
2	Delivery	M	14	07:05.858718	2	Clinic	M	14	05:57.009310
2	Delivery	M	6	07:39.300214	2	Clinic	M	5	06:31.365498
2	Delivery	M	2	08:02.570996	2	Clinic	M	2	06:59.967490
1	Flights	M	1	08:38.698360	2	Restaurant	C	2	07:18.927029
2	Flights	C	2	08:42.389884	2	Clinic	M	6	07:45.403075
1	Flights	M	16	10:18.809487	2	Restaurant	C	10	08:56.765163
1	Delivery	C	13	10:25.359836	1	Clinic	C	16	09:20.284593
2	Flights	C	9	10:51.761493	1	Restaurant	M	7	09:23.574403
1	Flights	M	7	10:52.183247	1	Restaurant	M	4	09:25.089084
2	Flights	C	10	10:52.495216	2	Restaurant	C	14	09:27.112225
1	Delivery	C	8	11:39.151434	2	Restaurant	C	3	09:55.736324
1	Delivery	C	15	12:03.519008	1	Clinic	C	15	10:25.475603
1	Flights	M	4	12:17.693128	2	Restaurant	C	5	10:37.460834
2	Flights	C	3	12:26.993837	2	Restaurant	C	9	10:49.014842
2	Flights	C	14	12:49.585392	1	Restaurant	M	16	10:56.743477
2	Flights	C	11	13:04.272941	1	Clinic	C	13	11:04.485390
1	Delivery	C	4	13:16.470523	1	Clinic	C	4	12:06.690347
1	Delivery	C	1	15:47.376327	1	Clinic	C	8	13:38.014602
1	Delivery	C	16	18:02.259692	1	Clinic	C	12	14:37.197260
1	Flights	M	12	20:03.920754	1	Restaurant	M	1	17:09.073104
2	Flights	C	5	21:32.272442	2	Restaurant	C	11	17:11.980052
2	Flights	C	6	23:10.727760	1	Clinic	C	7	19:35.816561
1	Delivery	C	7	23:20.991158	2	Restaurant	C	6	28:02.391335
1	Delivery	C	12	41:29.414342	1	Clinic	C	1	28:18.301114

The timings table contains data captured for each study, in this case, the maintenance study. By analysing this table, it is possible to identify the time of each participant, their group, the technique and the considered application.

We have developed a data collector to gather the experiment data. This system has stored the timings with milliseconds precision considering both the server and clients' system clocks. However, the values presented in this paper only consider the server time. The delays of transmission by the computers are not taken into consideration; preliminary calculations considering the clients' clocks have indicated that these delays are insignificant, i.e., have not changed the hypothesis testing results. The server's clock was considered because we could verify that its clock had not been changed throughtout the execution.

That system was able to gather the timings and the supplied information transparently. The participants only had to execute the start time, which was supervised, and to work on the processes independently. After the test case had provided successful results, which meant that the framework was correctly coupled, the finish time was automatically submitted to the server before notifying the success to the participant.

5.4 Data analysis and interpretation for reuse study

The data of the first study is found on Table 11, which is arranged by the time taken to complete the process. The first noteworthy information found on this table is that the model-based reuse tool, which is identified by the letter 'M', is found in the first twelve results. The conventional process, which is identified by the letter 'C', got the last four results.

The timings data of Table 11 is also represented graphically in a bar graph, which is plotted on Figure 10. The same identifying code for each participant and the elapsed time in seconds are visible on the graph. The bars for the used conventional technique and the used model tool are paired for each participant, allowing easier visualization of the amount of time taken by each of them. In other words, the taller the bar, the more time it took to complete the process with the specified technique.

The second significant information found during the first study was that not a single participant could reuse the framework faster by using the conventional process than by using the reuse tool in the same activity.

Table 12 shows the average timings and their proportions. If we analyze the average time that the participants from both groups have taken to complete the processes, we could conclude that the conventional technique took approximately 97.64% longer than the model-based tool.

5.5 Maintenance study definition

It is necessary to remind here that our objective was to compare the effort in modifying a CF-based application by editing the reuse code (conventional technique) with the effort in modifying the same application by editing the RM. The Persistence CF, shown in the Section 3.3 was again used in the two maintenance exercises. The quantitative focus was measured by means of the time spent in the maintenance tasks and the qualitative focus was to determine which artifact (source code or RM) takes less effort during maintenance. This experiment was conducted from the perspective of application engineers

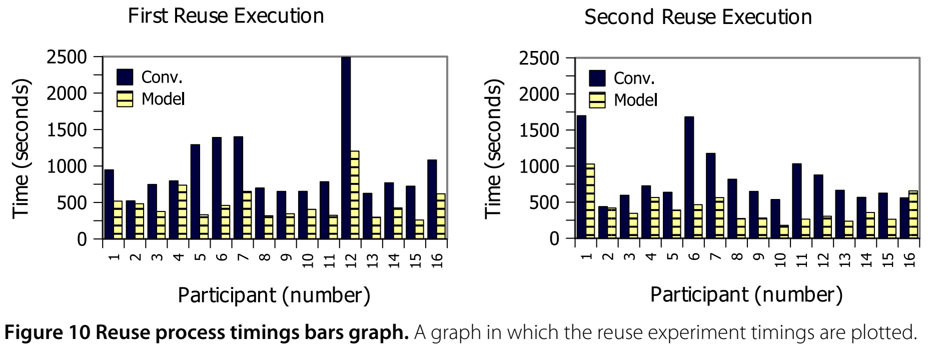


Figure 10 Reuse process timings bars graph. A graph in which the reuse experiment timings are plotted. Each bar represents the time taken by each participant in order to complete the reuse activities, which is identifiable by the participant number.

Table 12 Reuse study average timings

A.	Tech.	Avg.	Sum of Avg.	Percents
First	Conv.	16:13.44008	30:03.79341	66.7766%
Replication		13:50.35333		
First	Model	08:04.980525	14:57.441176	33.2234%
Replication		06:52.460651		
Total			45:01.234586	100.0000%

The Study Average Timings table contains averages for the Reuse Study. It is possible to compare the general time effort needed to complete the activities of both techniques.

who intended to maintain CF-based applications. Therefore, the study object is the 'effort' of maintaining a CF-based application.

5.6 Maintenance study planning

The core question we wanted to answer here was: "Which artifact takes less editing effort during maintenance: the reuse model or the reuse code?" During this experiment we have gathered and analyzed the timings taken to complete the process for each activity.

5.6.1 Context selection

Both studies were conducted by students of the Computer Science Department. In this section, they are referred to as "participants". Sixteen participants took part in the experiments: eight of them were undergraduate students and the other eight were graduate students. Every participant had a prior AspectJ experience.

5.6.2 Formulation of hypotheses

Table 13 contains three variables. "Tc_m" represents the overall time to edit the reuse code during maintenance. "Tm_m" represents the overall time to edit the reuse model during maintenance. "$H0_m$" represents the null hypothesis, which is true when the edition of both artifacts is equivalent. "Hp_m" represents the first alternate hypothesis, which is true when the edition of the reuse code takes longer than editing the RM. "Hn_m" represents the second alternate hypothesis, which is true when the edition of the reuse code takes less time than editing the RM. These hypotheses are also mutually exclusive: only one of them is true.

Table 13 Hypotheses for the maintenance study

H0$_m$	There is no difference between using editing a reuse model and editing the reuse code in terms of productivity (time) when maintaining an application that reuses a CF. Then, it is equivalent to edit any of the artifacts. Tc_m - $Tm_m \approx 0$
Hp$_m$	There is a positive difference between using editing a reuse model and editing the reuse code in terms of productivity (time) when maintaining an application that reuses a CF. Then, editing the reuse code takes more time than editing a reuse model during maintenance. Tc_m - $Tm_m > 0$
Hn$_m$	There is a negative difference between using editing a reuse model and editing the reuse code in terms of productivity (time) when maintaining an application that reuses a CF. Then, editing the reuse code takes less time than editing a reuse model during maintenance. Tc_m - $Tm_m < 0$

Considering the Reuse Study, there are three hypotheses for the outcome. In the first one, both are equivalent, while in two of them, a technique is faster.

5.6.3 Variable Selection

The dependent variable analyzed here was the "time spent to complete the process". The independent variables, which were controlled and manipulated, are: "Base Application", "Technique" and "Execution Types".

5.6.4 Participant selection criteria

The participants were selected through a non probabilistic approach by convenience, i. e., the probability of all population elements belong to the same sample is unknown. Both studies share the same participants.

5.6.5 Design of the Maintenance Study

The participants were divided into two groups. Each group was composed of four graduate students and four undergraduate students. Each group was also balanced considering the characterization form of each participant and their results from the first study. The phases for this study are shown in Table 14.

5.6.6 Instrumentation for the maintenance study

The base applications provided for the second study were modified versions of the same applications that had been supplied during the first study. These applicationswere provided with incorrect reuse codes (conventional) and incorrect reuse models (model-based): these incorrect artifacts had to be fixed by the participants. The participants received a document describing possible generic errors that could happen when a reuse code or a model are defined incorrectly. It is important to point out that that document did not have details regarding the base applications; the participants had to find the errors by browsing the source code.

The provided applications had the same reuse complexity: the reuse codes and models had the same amount of errors. In order to fix each CF coupling, the participants had to fix three outdated class names, three outdated method names, and three mistyped characters. It is also important to emphasize that errors specific for the manual edition of reuse codes were not inserted in this study.

Each phase row of the Table 14 is divided into sub-rows that contain the name of the application and the technique employed during the maintenance. For instance, the participants of the first group had to fix the reuse code of the "Deliveries Application" during

Table 14 Maintenance study design

Phase	Group 1	Group 2
General training	Maintenance training	
	Deliveries application	
2nd First	Models	Conventional
Maintenance phase	Flights application	
1st Replication	Conventional	Models
Maintenance phase	Medical clinic application	
2nd Replication	Models	Conventional
Maintenance phase	Restaurant application	

The Study Design contains every phase from both studies. It contains the sequence of operations, technique and the conidered applications.

the First Maintenance Phase, while the participants of the second group had to fix the reuse model to perform the same exercise.

5.7 Operation for maintenance study

5.7.1 Preparation

During the maintenance study, the students had to fix a reuse artifact to complete the process. Every participant had to fix every application. They have fixed each application only once, by using only one of the techniques in equal numbers.

5.7.2 Operation Execution

The participants had to work with two applications; each group started with a different technique. The secondary executions were replications of the primary executions with two other applications They were created to avoid the risk of getting unbalanced results during the primary execution, since some data that we gathered during the pilot study were rendered invalid.

5.7.3 Data validation

The forms filled by the participants were confirmed with the preliminary data gathered during the pilot study. In order to provide a better controllability, the researchers also watched the notifications from the data collector to check if the participants had concluded the maintenance process and had gathered the necessary data.

5.7.4 Data collection

The timings for the maintenance study are presented in Table 15. The column "G" stands for the group of the participant; "A" stands for the application being reused; "T" stands for the reuse technique which is either "C" for conventional or "M" for model-based tool; "Time" column lists the time the participant spent to complete each phase, and finally; and "P" column lists an identifying code of the participants. At least eight values are allocated to graduate students and the rest are undergraduate students; The raw data we have gathered during the maintenance study is also available as Additional file 2.

The data collector that was employed to gather the experiment data stored the timings with milliseconds precision: both the server and clients' system clocks were taken into consideration. However, the values presented in this paper consider only the server time. The delay of data transmission over the network was not taken into consideration. We believe that they are insignificant in this case because preliminary calculations considering the clients' clocks did not change the order of results.

That system was able to gather the timings and the supplied information transparently. The unique task of the participants was to click in a button to initialize the starting time. Once the provided test case had succeed (meaning that the framework was correctly coupled) the finishing time was automatically submitted to the server before notifying the success to the participant.

5.8 Data analysis and interpretation for maintenance study

The data of the second study is found on Table 15. This study has provided results similar to the first study. The first eleven values are related to the model-based tool, while the last

Table 15 Maintenance process timings

Primary execution					Secondary execution				
G	**A**	**T**	**P**	**Time**	**G**	**A**	**T**	**P**	**Time**
2	Flights	C	10	02:30.944685	2	Clinic	M	5	01:43.801965
2	Flights	C	9	02:54.232578	2	Clinic	M	3	02:17.158954
1	Delivery	C	8	03:02.751342	1	Clinic	C	8	02:34.248260
2	Flights	C	2	03:11.695431	1	Clinic	C	14	02:57.405545
1	Delivery	C	15	03:31.801582	2	Restaurant	C	2	03:01.547524
2	Flights	C	12	03:45.692316	2	Restaurant	C	10	03:09.169865
2	Flights	C	3	05:09.817914	2	Clinic	M	2	03:25.640129
2	Flights	C	5	05:44.462030	2	Restaurant	C	3	03:39.443080
1	Flights	M	8	05:53.407296	1	Clinic	C	7	04:28.998071
2	Flights	C	11	07:08.687074	2	Restaurant	C	6	04:35.517498
2	Flights	C	6	07:38.576312	2	Restaurant	C	12	04:41.052812
1	Flights	M	4	07:53.595699	2	Restaurant	C	11	04:46.028085
1	Flights	M	14	08:14.148937	1	Restaurant	M	8	04:51.290971
2	Delivery	M	3	08:27.092566	2	Clinic	M	6	04:53.800449
1	Delivery	C	1	08:37.138931	1	Restaurant	M	15	04:58.094389
1	Flights	M	13	08:50.185469	1	Clinic	C	15	05:21.846560
1	Flights	M	1	09:15.253791	2	Restaurant	C	5	05:42.389865
2	Delivery	M	5	09:15.934211	2	Clinic	M	10	07:18.533351
1	Delivery	C	14	09:32.031612	1	Restaurant	M	14	07:24.342788
1	Delivery	C	7	10:04.694800	1	Clinic	C	16	07:37.332151
1	Flights	M	15	11:07.617639	1	Restaurant	M	1	07:44.516376
2	Delivery	M	6	11:32.482992	2	Clinic	M	11	08:08.144168
2	Delivery	M	2	11:49.247460	2	Restaurant	C	9	08:13.115942
1	Delivery	C	16	12:12.576158	1	Restaurant	M	13	08:32.056119
1	Flights	M	7	12:27.297563	1	Restaurant	M	16	11:28.592180
1	Delivery	C	13	12:49.443610	1	Restaurant	M	7	11:45.459699
2	Delivery	M	11	13:00.604583	2	Clinic	M	9	12:42.958789
1	Delivery	C	4	13:25.433748	1	Restaurant	M	4	13:57.879299
2	Delivery	M	9	15:51.117061	1	Clinic	C	1	14:46.465482
2	Delivery	M	12	15:56.048486	1	Clinic	C	4	17:55.176353
2	Delivery	M	10	21:23.533192	1	Clinic	C	13	18:02.486509
1	Flights	M	16	32:32.875079	2	Clinic	M	12	25:54.176697

The timings table contains data captured for each study, in this case, the maintenance study. By analysing this table, it is possible to identify the time of each participant, their group, the technique and the considered application.

four are related to the conventional technique. Only the Participant 16 was able to reuse the framework faster by applying the conventional process, which contradicts the results of the same participant in the previous study. This participant said he got confused when he had to correct the reuse model. That was the reason why he had to restart the process from the very beginning, causing this longer time.

The plots for the maintenance study are found on Figure 11. These plots follow the same guidelines that were used when plotting the graphs for the previous study. Considering the timings of the maintenance study, the reuse model edition does not provide any advantage in terms of productivity, since most of participants took longer to edit the model than the reuse code.

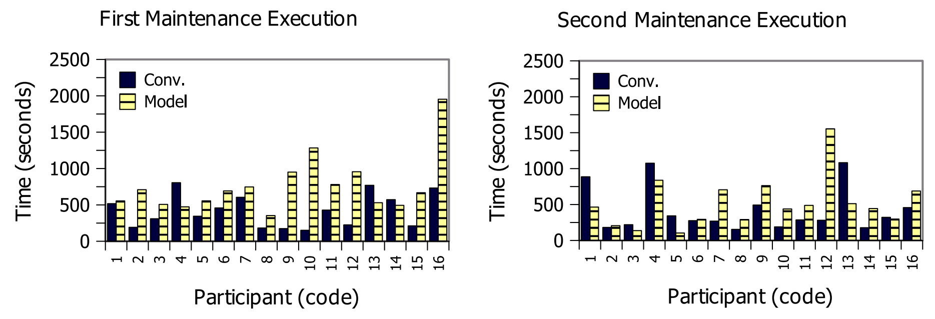


Figure 11 Maintenance process timings bars graph. A graph in which the maintenance experiment timings are plotted. Each bar represents the time taken by each participant in order to complete the maintenance activities, which is identifiable by the participant number.

Table 16 illustrates the average timings and their proportions. Considering only the average time, the participants who applied the conventional technique took less time than their counterparts who used our model-based approach.

6 Results and discussion

6.1 Hypotheses testing for reuse study

In this section, we present statistical calculations to evaluate the data of the reuse study. We applied Paired T-Tests for each execution and another T-Test after removing eight outliers. The time consumed in each execution was processed using the statistic computation environment "R" (Free Software Foundation, Inc 2012). The results of the T-Tests are shown on Table 17, which is actually a pair of tables. The time unit is "seconds".

The first columns of these tables contain the type of T-Test and the second ones indicate the source of the data. The "Means" columns indicate the resultant mean for each T-Test. For a paired T-Test, there is one mean, which is the average of subtracting each set member by its counterpart in the other set. For the non-paired T-Tests, there are two means, which are the averages for each set. In this case, the first set represents the conventional technique; the second set represents the use of the model-based tool. The "d.t." columns stand for the degrees of freedom, which is related to how many different values are found in the sets; "t"and "p" are variables considered in the hypothesis testing.

The Paired T-Test is used to compare the differences between two samples related to each participant. In this case, the time difference of every participant is considered individually; then, the means of the differences are calculated. In the "Two-Sided" T-Tests, which are unpaired, the means are calculated for the entire group, because a participant

Table 16 Maintenance study average timings

A.	Tech.	Avg.	Sum of Avg.	Percents
First	Conv.	06:57.498758	13:55.762733	39.5521%
Replication		06:58.263975		
First	Model	12:43.152626	21:17.305521	60.4479%
Replication		08:34.152895		
Total		35:13.068254		100.0000%

The Study Average Timings table contains averages for the Maintenance Study. It is possible to compare the general time effort needed to complete the activities of both techniques.

Table 17 Reuse study t-test results

T-Test	Data	Means	d.f.	t	p
Paired	First	488.4596	15	5.841634	$3.243855 \cdot 10^{-05}$
Paired	Replication	417.8927	15	5.285366	$9.156136 \cdot 10^{-05}$
Two-sided	Both	771.4236 / 409.4295	43.70626	6.977408	$1.276575 \cdot 10^{-08}$

T-Test is a statistical test used to determine the correct hypothesis for both studies. This table lists the results for the reuse study.

may be an outlier in a specific technique, which breaks the pairs. It is referred to as two-sided because the two sets have the same number of elements, since the same number of outliers was removed from each group.

The “Chi-squared test” was applied to both studies in order to detect the outliers, which were then removed when calculating the unpaired T-Test. On the table, the unpaired T-Tests are refered as “Two-sided”. The results of the “Chi-squared test” for the reuse study are found on Table 18. The ‘M’ in the techniques column indicates the use of our tool, while ‘C’ indicates the conventional technique. The group column indicates the number of the group. The X^2 indicates the result of subtracting each value by the variance of the complete set. The position column indicates their position on the set, i.e., highest or lowest. The outlier column shows the timings in seconds that were considered abnormal.

In order to achieve a better visualization of the outliers, we also provide two plots of the data sets. In Figure 12 there are line graphs which may be used to visualize the dispersion of the timing records. In these plots, the timings for each technique are ordered by their performance; therefore, the participant numbers in these plots are not related to their identification codes.

Considering the reuse study and according to the analysis from Table 17 we can state the following. Since all p-values are less than the margin of error (0.01%), which corresponds to the established significance level of 99.99%, then, statistically, we can reject the “$H0_r$” hypothesis that states the techniques are equivalent. Since every t-value is positive, we can accept the “Hp_r” hypothesis, which implies that the conventional technique takes more time than ours.

6.2 Hypotheses testing for maintenance study

In this section, we present statistical calculations to evaluate the data of the maintenance study. Similarly to the reuse study, we applied Paired T-Tests for each execution

Table 18 Chi-squared test for outlier detection applied on reuse study

Study	T.	G.	X^2	p	Position	Outlier
First	C	1	5.104305	0.02386654	highest	2489.414342
		2	2.930583	0.08691612	highest	1390.72776
	M	1	4.091151	0.04310829	highest	1203.920754
		2	2.228028	0.1355267	highest	482.570996
Replication	C	1	4.552248	0.03287556	highest	1698.301114
		2	5.013908	0.02514448	highest	1682.391335
	M	1	3.917559	0.04778423	highest	1029.073104
		2	2.943313	0.08623369	lowest	179.467569

Chi-squared test is a statistical test used to detect outliers. It was employed to detect eight outliers in the reuse study. These outliers are removed in the last t-test.

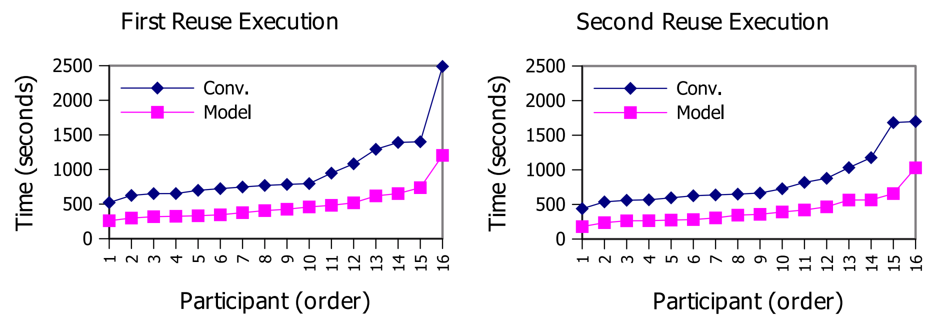


Figure 12 Reuse process timings line graph. A graph in which the reuse experiment timings are plotted. The timings are plotted as points in ascendant order, which are then linked with lines. This graph is useful to visualize the data dispersion.

and another T-Test after removing eight outliers. The seconds that were spent during the process were processed using the statistic computation environment "R" (Free Software Foundation, Inc 2012). The results of the T-Tests are shown on Table 19.

The first column of this table contain the type of T-Test. The second columns indicate the source of the data, which refers to the datasets created for each technique. The "Means" columns indicate the resultant means. The "d.t." columns stand for the degree of freedom; "t" and "p" are variables considered in the hypothesis testing.

The "Chi-squared test" was applied in order to detect the outlier. The results of the "Chi-squared test" for the maintenance study are found on Table 20. These outliers were removed when calculating the unpaired T-Test. On the table, the unpaired T-Test is refered as "Two-sided". The 'M' in the *techniques* column indicates the use of our tool, while 'C' indicates the conventional technique. The *group* column indicates the number of the group. The X^2 indicates the results of an comparison to the variance of the complete set. The position column indicates their position on the set, i.e., highest or lowest. And finally, the outlier column shows the timings in seconds that were considered abnormal.

In order to achieve better visualization of the outliers, we also provide two plots of the data sets. In Figure 13, there are line graphs which may be used to visualize the dispersion of the timing records. In these plots, the timings for each technique are ordered independently. Therefore, the participant numbers in these plots are not related to their identification codes.

If we take into consideration the maintenance study and its analysis illustrated on Table 19, we cannot reject the "$H0_m$" hypothesis that states the techniques are equivalent because all p-values are bigger than the margin of error (0.01%), which corresponds to the established significance level of 99.99%. Therefore, statistically , we can assume that the effort needed to edit a reuse code and a reuse model is approximately equal.

Table 19 Maintenance study t-test results

T-test	Data	Means	d.f.	t	p
Paired	First	-345.6539	15	-3.971923	0.001227479
Paired	Replication	-95.88892	15	-1.191781	0.2518624
Two-sided	Both	431.3323 641.0024	24.22097	-2.662684	0.0135614

T-Test is a statistical test used to determine the correct hypothesis for both studies. This table lists the results for the maintenance study.

Table 20 Chi-squared test for outlier detection applied on maintenance study

Study	T.	G.	X^2	p	Position	Outlier
First	C	1	2.350449	0.1252469	lowest	182.751342
		2	2.152789	0.1423112	highest	458.576312
	M	1	5.788559	0.0161308	highest	1952.875079
		2	3.598538	0.05783041	highest	1283.533192
Replication	C	1	1.771974	0.183138	highest	1082.486509
		2	4.338041	0.03726978	highest	493.115942
	M	1	2.422232	0.1196244	highest	837.879299
		2	4.87366	0.02726961	lowest	1554.176697

Chi-squared test is a statistical test used to detect outliers. It was employed to detect eight outliers in the maintenance study. These outliers are removed in the last t-test.

6.3 Threats to validity

6.3.1 Internal validity

- Experience Level of Participants. The different levels of knowledge of the participants could have compromised the data. To mitigate this threat, we divided the participants in two balanced groups considering their experience level and later we rebalanced the groups considering the preliminary results. Although all participants already had a prior experience in how to reuse the CF in the conventional way, during the training phase, they were taught how to make the reuse with the model-based tool and also how to reuse it in the normal way. So, this could have provided the participants even more experience with the conventional technique.
- Productivity under evaluation. The students could have thought that their results in the experiment will influence their grades in the course. In order to mitigate this, we explained to the students that no one was being evaluated and their participation was considered anonymous.
- Facilities used during the study. Different computers and configurations could have affected the recorded timings. However, participants used the same configuration, make, model, and operating system in equal numbers. The participants were not allowed to change their computers during the same activity. This means thatevery participant had to execute every exercise using the same computer.

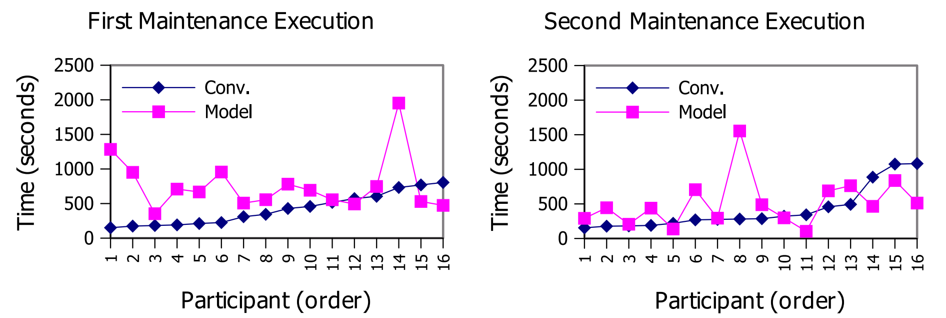


Figure 13 Maintenance process timings line graph. A graph in which the maintenance experiment timings are plotted. The timings are plotted as points in ascendant order, which are then linked with lines. This graph is useful to visualize the data dispersion.

6.3.2 Validity by construction

- Hypothesis expectations: the participants already knew the researchers and knew that the model-based tool was supposed to ease the reuse process, which reflects one of our hypothesis. Both of these issues could affect the collected data and cause the experiment to be less impartial. In order to avoid impartiality, we enforced that the participants had to keep a steady pace during the whole study.

6.3.3 External validity

- Interaction between configuration and treatment. It is possible that the reuse exercises were not accurate for every reuse of a crosscutting framework for real world applications. Only a single crosscutting framework was considered in our study and the base applications were of the same complexity. To mitigate this threat, we designed the exercises with applications that were based on the ones existing in reality.

6.3.4 Conclusion validity

- Measure reliability. It refers to metrics used to measuring the reuse effort. To mitigate this threat, we have used only the time, necessary to complete the process, which was captured by a data collector in order to allow better precision;
- Low statistic power. The ability of a statistic test to reveal the reliable data. We applied three T-Tests to analyze the experiment data statistically to avoid the low statistic power.

7 Related work

The approach proposed by Cechticky et al. (2003) allows the reuse of object-oriented application framework by applying the tool called OBS Instantiation Environment. That tool supports graphical models to define the settings to generate the expected application. The model-to-code transformation generates a new application that reuses the framework.

In another related work, Braga and Masiero (2003) proposed a process to create framework instantiation tools. The process is specific for application frameworks defined by a pattern language. The process application assures that the tool is capable of generating every possible framework variability.

The approach defined by Czarnecki et al. (2006) defines a round-trip process to create domain specific languages for framework documentation. These languages can be employed to represent the information that the framework programming interfaces need during the instantiation and the description of these interfaces. This is a bilateral process, i.e., two transformers are fashioned: a transformer from model to code and a transformer from code to model. Generated codes are transformed back to models. This allow the comparison of the source model with the generated model, which should be perfectly equal. If differences are found, the language or the transformers should be improved in the activity called “conciliation”.

Santos et al. proposed a process and a tool to suport framework reuse (Santos et al. 2008). In this approach, the domain engineer must supply a reuse example which must be tagged. These tags mark points of the reuse example that is to be replaced in order to create different applications.

The tags are mapped into a domain specific language that lists information that the application engineer should supply in order to reuse the framework. This domain specific language instances are, then, interpreted by a tool that is capable of listing the points to complete framework instantiation. This is the only related work that uses AspectJ and the aspect-oriented programming.

Our proposal differs from their approach on the following topics: 1) their approach is restricted to frameworks known during the development of the tool; 2) the reuse process is applied to application frameworks, which are used to create new applications.

Another approach was proposed by Oliveira et al. (2011). Their approach can be applied to a greater number of object oriented frameworks. After the framework development, its developer may use the approach to ease the reuse by writing the cookbook in a formal language known as Reuse Definition Language (RDL) which can also be used to generate the source code. This process allows us to select variabilities and resources during the reuse procedure, as long as the framework engineer specifies the RDL code correctly. These approaches were created to support the reuse procedure during the final development stages. Therefore, the approach that is proposed in this paper differs from others by supporting earlier development phases. This allows the application engineer to initiate the reuse process since the analysis phase while developing an application compatible to the reused frameworks.

Although the approach proposed by Cechticky et al. (2003) is specific for only one framework, it can be employed since the design phase. The other related approach can be employed in a greater number of frameworks: however, it is used on a lower abstraction level, and does not support the design phase. Another difference is the generation of aspect-oriented code, which improves code modularization. Finally, the last difference that must be pointed out is the use of experiments to evaluate the approach, while the presented related works only show case studies employing their tools.

8 Conclusions

In this article we presented a model-based approach that raises the abstraction level of reusing Crosscutting Frameworks (CFs) - a type of aspect-oriented framework. The approach is supported by two models, called Reuse Requirements Model (RRM) and Reuse Model (RM). The RRM serves as a graphical view for enhancing cookbooks and the RM supports application engineers in performing the reuse process by filling in this model and generating the source-code. Considering our approach, a new reuse process is delineated allowing engineers to start the reuse in early development phases. Using our approach, application developers do not need to worry about either architectural or source-code details, shortening the time necessary to conduct the process.

We have evaluated our approach by means of two experiments. The first one was focused on comparing the productivity of our model-based approach to the ad-hoc approach. The results showed the improvement of approximately 97% in favor of our approach. We claim that this improvement can be influenced by the framework characteristics but not by the application characteristics. If a CF requires a lot of heterogeneous joinpoints we think this percentage will go down because the application engineer will need to write the joinpoints (method names, for instance) either using both our approach or the ad-hoc one. However, if the CF is heavily based on inter-type declarations and the returning of values, then we claim that the productivity can be even higher, as it is very straightforward to do so while using our approach.

The second experiment was focused on observing the effort in maintaining applications that were developed with our approach (CF-based applications) and with the ad-hoc one. It was not possible to conclude which process takes less effort in this case; however, we believe that they are approximately equivalent. The participants argued that the tool could be improved to avoid opening new forms while entering the model attributes, which, as they claim, had disrupted their work and prevented them from reaching a better performance in this case. It is important that in this experiment we did not provide errors that developers could create while using the conventional approach, since our model approach shields the developers from doing that. We have also provided the raw data gathered during the studies as Additional files 1 and 2.

As the possible limitations of our work, we can mention the following. Once the models have been created on top of the Eclipse Modeling Project, they cannot be used in another development environment. Besides, the code generator only produce codes for AspectJ, therefore, only crosscutting frameworks developed in this language can be currently supported. A simple extension is possible to allow this approach to support the reuse of non-crosscutting frameworks written in Java and AspectJ. Also, we have not yet evaluated how to deal with coupling of multiple CFs to a single base application. Although this functionality is already supported in our approach, some frameworks may select the same joinpoints, which may cause conflicts and lead to unpredictable results.

Long term future works are: (i) providing a support for framework engineers so that they do not have to build the RRM manually. The idea is to develop a tool which can assist them in creating this model in a more automatic way; (ii) performing an experiment to verify whether the abstractions of the model elements are on a suitable level (iii) analyzing the reusability of the metamodel abstract classes.

Additional files

Additional file 1: Contains the raw data gathered during the Reuse study.
Additional file 2: Contains the raw data gathered during the Maintenance study.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
Every author was important for the completion of this article, however, their previous activities are also important to reach the research results, then, these activities are listed in this section. TG developed the models, model editors and model transformers. He also designed and conducted the studies and the considered applications. This work was also presented as a master's thesis in Federal University of São Carlos, Brazil. RSD was a contributor who developed the related tools in which this work is related to. He developed a repository for crosscutting frameworks that allow their sharing and integrates to the tool described herein in order to provide a full featured crosscutting framework reuse environment. These tools are available as Eclipse plugins. VVdC is a professor at Federal University of São Carlos which is responsible for the crosscutting framework reuse project. He also developed the crosscutting framework used as example and worked on the study executions. OPL is a professor at Polytechnic University of Valencia that provided useful background information regarding model-driven development and code generator tools. He is also part of the crosscutting framework reuse project. All authors read and approved the final manuscript.

Acknowledgements
The authors would like to thank CNPq for funding (Processes 132996/2010-3 and 560241/2010-0) and for the Universal Project (Process Number 483106/2009-7) in which this article was created. Thiago Gottardi would also like to thank FAPESP (Process 2011/04064-8).

Author details
[1]Departmento de Computação, Universidade Federal de São Carlos, Caixa Postal 676, 13.565-905, São Carlos, São Paulo, Brazil. [2]Instituto de Ciências Matemáticas e Computação, Universidade de São Paulo, Av. Trabalhador São Carlense, 400, São Carlos, São Paulo, Brazil. [3]Universidad Politecnica de Valencia, Camino de Vera s/n, Valencia, Spain.

Received: 7 February 2013 Accepted: 6 September 2013
Published: 29 October 2013

References

Antkiewicz M, Czarnecki K (2006) Framework-specific modeling languages with round-trip engineering. In: ACM/IEEE 9th international conference on model driven engineering languages and systems (MoDELS). Springer-Verlag, Genova, pp 692–706. http://www.springerlink.com/content/y08152212701l160/fulltext.pdf

Apache Software Foundation (2012) Apache Derby. http://db.apache.org/derby/

AspectJ Team (2003) The AspectJ(TM) Programming Guide. http://www.eclipse.org/aspectj/doc/released/progguide/

Braga R, Masiero P (2003) Building a wizard for framework instantiation based on a pattern language. In: Konstantas D, Léonard M, Pigneur Y, Patel S (eds) Object-oriented information systems, Volume 2817 of Lecture notes in computer science. Springer, Berlin / Heidelberg, pp 95–106. http://dx.doi.org/10.1007/978-3-540-45242-3_10

Bynens M, Landuyt D, Truyen E, Joosen W (2010) Towards reusable aspects: the mismatch problem. In: Workshop on Aspect, Components and Patterns for Infrastructure Software (ACP4IS'10). Rennes and Saint Malo, France. ACM, New York, NY, USA, pp 17–20

Camargo VV, Masiero PC (2005) Frameworks Orientados A Aspectos. In: Anais Do 19° Simpósio Brasileiro De Engenharia De Software (SBES'2005). Uberlândia-MG, Brasil, Outubro

Camargo VV, Masiero PC (2004) An approach to design crosscutting framework families. In: Proc. of the 2008 AOSD workshop on Aspects, components, and patterns for infrastructure software, ACP4IS '08. Brussels, Belgium. ACM, New York, NY, USA. http://dl.acm.org/citation.cfm?id=1404891.1404894

Cechticky V, Chevalley P, Pasetti A, Schaufelberger W (2003) A generative approach to framework instantiation. In: Proceedings of the 2nd international conference on Generative programming and component engineering, GPCE '03. Springer-Verlag, New York, Inc., New York, pp 267–286. http://portal.acm.org/citation.cfm?id=954186.954203

Clark T, Evans A, Sammut P, Willans J (2004) Transformation Language Design: A Metamodelling Foundation, ICGT, Volume 3256 of Graph Transformations, Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, pp 13–21

Clements P, Northrop L (2002) Software product lines: practices and patterns, 3rd edn. The SEI series in software engineering, 563 pages, first edition. Addison-Wesley Professional, Boston, United States of America. http://www.pearsonhighered.com/educator/product/Software-Product-Lines-Practices-and-Patterns/9780201703320.page

Cunha C, Sobral J, Monteiro M (2006) Reusable aspect-oriented implementations of concurrency patterns and mechanisms. In: Aspect-Oriented Software Development Conference (AOSD'06). Bonn, Germany. ACM, New York, NY, USA

Eclipse Consortium (2011) Graphical Modeling Framework, version 1.5.0. Graphical Modeling Project. http://www.eclipse.org/modeling/gmp/

Efftinge S (2006) openArchitectureWare 4.1 Xtend language reference. http://www.openarchitectureware.org/pub/documentation/4.3.1/html/contents/core_reference.html

Fayad M, Schmidt DC (1997) Object-oriented application frameworks. Commun ACM 40: 32–38

Fowler M (2010) Domain specific languages, 1st edition. 640 pages, first edition. Addison-Wesley Professional, Boston, United States of America. http://www.pearsonhighered.com/educator/product/DomainSpecific-Languages/9780321712943.page

France R, Rumpe B (2007) Model-driven development of complex software: a research roadmap. In: 2007 Future of Software Engineering, FOSE 07. IEEE Computer Society, Washington, pp 37–54

Free Software Foundation, Inc (2012) R. http://www.r-project.org/

Huang M, Wang C, Zhang L (2004) Towards a reusable and generic aspect library. In: Workshop of the Aspect Oriented Software Development Conference at AOSDSEC'04, AOSD'04. Lancaster, United Kingdom. ACM, New York, NY, USA

Kiczales G, Lamping J, Mendhekar A, Maeda C, Lopes C, marc Loingtier J, Irwin J (1997) Aspect-oriented programming. In: ECOOP. Springer-Verlag, Heidelberger, Berlin, Germany

Kiczales G, Hilsdale E, Hugunin J, Kersten M, Palm J, Griswold WG (2001) An overview of aspectJ. Springer-Verlag, Heidelberger, Berlin, Germany, pp 327–353

Kulesza U, Alves E, Garcia R, Lucena CJPD, Borba P (2006) Improving Extensibility of object-oriented frameworks with aspect-oriented programming. In: Proc. of the 9th Intl Conf. on software reuse (ICSR'06). Torino, Italy, June 12-15, 2006, Lecture Notes in Computer Science, Programming and Software Engineering, vol 4039. Springer-Verlag, Heidelberger, Berlin, Germany, pp 231–245

Mortensen M, Ghosh S (2006) Creating pluggable and reusable non-functional aspects in AspectC++. In: Proceedings of the fifth AOSD workshop on aspects, components, and patterns for infrastructure Software. Bonn, Germany. ACM, New York, NY, USA

Oliveira TC, Alencar P, Cowan D (2011) ReuseTool-An extensible tool support for object-oriented framework reuse. J Syst Softw 84(12): 2234–2252. http://dx.doi.org/10.1016/j.jss.2011.06.030

Pastor O, Molina JC (2007) Model-driven architecture in practice: a software production environment based on conceptual modeling. Springer-Verlag, New York, Secaucus

Sakenou D, Mehner K, Herrmann S, Sudhof H (2006) Patterns for re-usable aspects in object teams. In: Net Object Days. Erfurt, Germany. Object Teams, Technische Universität Berlin, Berlin, Germany

Santos AL, Koskimies K, Lopes A (2008) Automated domain-specific modeling languages for generating framework-based applications. Softw Product Line Conf Int 0: 149–158

Schmidt DC (2006) Model-driven engineering. IEEE Computer 39(2). http://www.truststc.org/pubs/30.html

Shah V, Hill V (2004) An aspect-oriented security framework: lessons learned. In: Workshop of the Aspect Oriented Software Development Conference at AOSDSEC'04, AOSD'04. Lancaster, United Kingdom. ACM, New York, NY, USA

Soares S, Laureano E, Borba P (2006) Distribution and persistence as aspects. Software: Practice and Experience 36(7): 711–759. John Wiley & Sons, Ltd. Hoboken, NJ, USA. http://onlinelibrary.wiley.com/doi/10.1002/spe.715/abstract

Soudarajan N, Khatchadourian R (2009) Specifying reusable aspects. In: Asian Workshop on Aspect-Oriented and Modular Software Development (AOAsia'09). Auckland, New Zealand. AOAsia, Chinese University of Hong Kong, Hong Kong, People's Republic of China

Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2000) Experimentation in software engineering: an introduction. First edition. 204 pages. Kluwer Academic Publishers, Norwell, MA, USA

Zanon I, Camargo VV, Penteado RAD (2010) Reestructuring an application framework with a persistence crosscutting framework. INFOCOMP 1: 9–16

doi:10.1186/2195-1721-1-4
Cite this article as: Gottardi *et al.*: **Model-based reuse for crosscutting frameworks: assessing reuse and maintenance effort.** *Journal of Software Engineering Research and Development* 2013 **1**:4.