Model-Based Reuse for Crosscutting Frameworks: Assessing Reuse and Maintainability Effort

Thiago Gottardi*, Rafael Serapilha Durelli†, Oscar Pastor López‡ and Valter Vieira de Camargo*
*Departamento de Computação, Universidade Federal de São Carlos,
Caixa Postal 676 – 13.565-905, São Carlos – SP – Brazil
Email: {thiago_gottardi,valter}@dc.ufscar.br
†Instituto de Ciências Matemáticas e Computação, Universidade de São Paulo,
Av. Trabalhador São Carlense, 400, São Carlos – SP – Brazil
Email: rsdurelli@icmc.usp.br
‡Universidad Politecnica de Valencia, Camino de Vera s/n, Valencia, Spain
Email: opastor@dsic.upv.es

***Abstract*—Over the last years a number of Crosscutting Frameworks (CFs) have been developed employing white-box strategies. This strategy requires significant technical knowledge to reuse these frameworks, such as, knowledge in specific programming languages, architectural details and also about the framework nomenclature. Besides, the reuse process can only be initiated when the development reaches the implementation phase, avoiding starting the reuse process in early development phases. In this paper we present a model-based technique for reusing CFs that improves the productivity by allowing the application engineer to concentrate on what is really important during the reuse process. We also present the foundations of our approach and also the result of two experiments that uses two versions of a Persistence CF; the original and the model-based. The results were promising regarding the effort employed to conduct the reuse process, but almost no difference was noticed concerning the effort in conducting maintenance activities.**

I. Introduction

Model-Driven Development (MDD) consists of the combination of generative programming, domain-specific languages and model transformations. MDD aims to reduce the semantic gap between the program domain and the implementation, using high-level models that shield software developers from complexities of the underlying implementation platform [?].

On the other hand, Aspect-Oriented Programming (AOP) is a programming paradigm that overcomes the limitations of Object-Orientation by providing abstractions able to modularize crosscutting concerns (CC) such as persistence, security and distribution. Among these abstractions, Pointcuts are expressions used to capture join-points of an application, e.g., method calls and executions and variable accesses. By capturing these join-points, it is possible to write code to be executed upon a Pointcut occurrence in a modular fashion. AspectJ is one of the AOP languages that implement these abstractions [?]. It is also the language employed in our work.

Since the advent of AOP, several researchers have investigated how its abstractions and concepts impact reuse methodologies, like product lines [?] and frameworks [?]. Many of these researchers investigated how to design a CC in a generic way to enhance their reusability [?], [?], [?], [?], [?], [?], [?], [?], [?], [?], [?], [?]. Several terms are used to represent this kind of design, e.g., reusable aspects [?], aspect-oriented frameworks and aspect libraries [?]. Because the absence of a taxonomy for this kind of design, we have defined and employed the term "Crosscutting Framework" (CF) to represent a specific kind of abstract aspect-oriented framework implementation of a single CC [?].

Most of the CFs found in literature apply white-box reuse strategies in their instantiation process, relying on writing source code to reuse the framework [?], [?], [?], [?], [?], [?], [?], [?], [?], [?], [?], [?]. White-box strategy makes application engineers to worry about low level implementation details during the reuse process, leading to the following problems: (*i*) to get know coding details regarding the programming paradigm employed in the framework, making the learning curve steeper; (*ii*) coding mistakes are more likely to happen when the reuse code is created manually; (*iii*) several lines of code must be written for the definitions of small number of hooks, impacting development productivity and (*iv*) reuse process can only be started during implementation phase, as there is no source code in earlier phases.

To overcome these problems, in this paper, we present an approach for supporting the reuse of CFs. The approach is based on two models: Reuse Requirements Model (RRM) and Reuse Model (RM). The RRM documents all the features and variabilities of a CF and is a partial replacement for cookbooks. Based on the RRM, the application engineer can then select just the desired features, building a more specific model, referred as RM. The application engineer has the opportunity to conduct the reuse process in our model-driven approach by filling the fields of this model, which is also used for code generation.

We also present the results of two experiments. In both experiments we have used the same Persistence CF [?]. Our approach showed benefits for the instantiation time, however, no differences were identified regarding the maintenance effort. Therefore, the main contributions of this paper are: (*i*) presenting a model-based approach for CFs, (*ii*) presenting the results of two experiments and (*iii*) the problems of the presented approach can be generalized to other model-based framework reuse processes.

In Section II the notion related to CF, their details and a CF's description that is used in this paper are showed; both the proposed approach and an example of instantiation related to a member of persistence CF are presented in Section III; in Section IV, an empirical evaluation is presented; in Section V, there are related works and in Section VI, there are the conclusions.

II. Crosscutting Frameworks

Crosscutting Frameworks (CF) encapsulate the generic behavior of a single crosscutting concern [?], [?], [?], [?]. There are CFs developed for persistence [?], [?], security [?], cryptography [?], distribution [?] and other concerns [?]. Their main objective is to make the reuse of such concerns easier during the development of an application.

As well as other types of frameworks, CFs also need information regarding the base application in order to be reused correctly and work properly. We named these 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) which are the roles played by users; 3) how many 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 in order to become functional. The standard process to reuse a CF is composed by two activities: instantiation and composition. The instantiation is when the application engineer is choosing variabilities and implementing hooks, while the coupling is when he/she is providing composition rules to couple the chosen variabilities to a base code. During the composition activity, pointcuts and composition rules are defined, unifying the chosen variabilities and the base code.

Applications developed with CFs are composed by three types of code modules: base, reuse and framework. The "base code" represents code of the base application. In the "framework code" there is the code of the CF, which is untouched during the reuse process. The "reuse module" is the connection between the base application and a framework. Each final application can be composed by several CFs, each one coupled by a reuse module. The code that was created specifically to reuse an CF is referred here as "reuse code" and applications which were developed based on CFs is referred here as CF-based Applications.

In a previous work we have developed a Persistence CF [?], which is used as case study in this paper. This CF has some features, for instance, both "Persistence" and "Connection" are mandatory features. The first one, aims to introduce a set of persistence operations (e.g., store, remove, update, etc) into applications persistence classes. The second feature, is related to the database connection and identifies points in the application code where the connection should be opened and closed. This feature has variabilities, as for example, the Database Management System (e.g., MySQL, SyBase, Native and Interbase). The CF also has a set of optional features such as "Caching", which is used to improve performance by keeping copies of the data in local memory, and "Pooling" which represents a number of active database connections.

III. Model-Based Reuse Approach

In order to assist the instantiation and composition of members of a CF we have put forward two new models, "Reuse Requirements Model" (RRM) and "Reuse Model" (RM). These models have been devised on top of Eclipse Modeling Framework and Graphical Modeling Framework [?] The formal definition of both models is specified by a single metamodel, which is shown in Figure 1. This metamodel is a set of enumerations and metaclasses, which are either concrete or abstract.

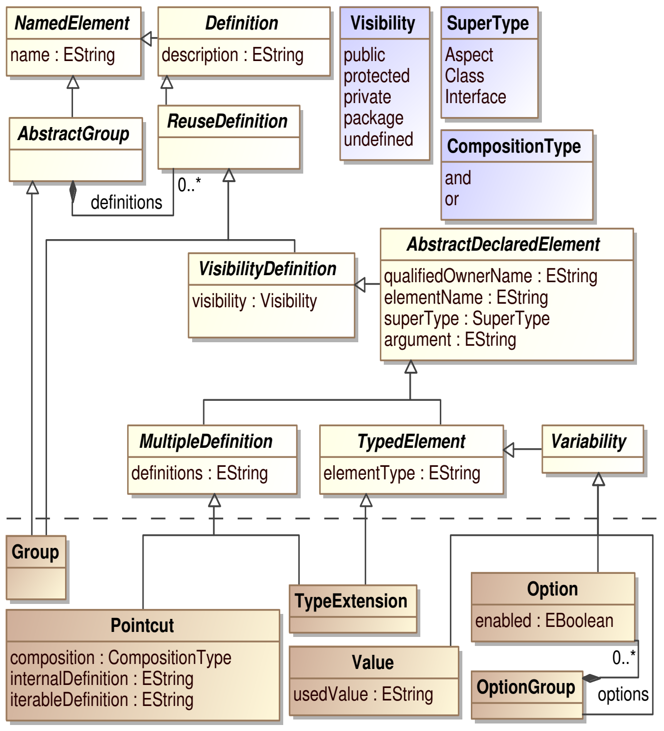


Fig. 1. Metamodel of the proposed models

The metamodel was built based on the vocabulary commonly used in the context of CFs. Among these concepts, there are pointcuts, classifier extensions, method overriding to return values and variabilities selections. These concepts were then mapped into concrete metaclasses, which are visible under the dashed line in Figure 1.

Above the dashed line in Figure 1, there are also the following enumerations: "Visibility", "SuperType" and "CompositionType", which are sets of literals used as properties of the metaclasses. The other elements above the line are abstract metaclasses. They were created after performing an analysis to generalize the properties of the concrete metaclasses. These abstract metaclasses can be applied in similar approaches and are also important to improve modularity and avoid code replication of the reuse code generator.

In Figure 2 there is an overview of our tool which is used to edit both of our proposed models. On the right of Figure 2, there is a "Palette" with the possible elements that can be inserted into these models. These elements are instances of the concrete metaclasses of the metamodel, visible under the dashed line in Figure 1. Their uses are: (*I*) "Group": an element to group any element visible in the models, including child groups; (*II*) "Pointcut": employed to override abstract pointcuts which represent join-points of the base application code that should be affected by the CF; (*III*) "TypeExtension": elements used to represent types found in the base application that must extend or implement classes, aspects or interfaces found in the CF; (*IV*) "Value": elements used to override methods to return any numeric or textual values that must be informed while reusing the CF; (*V*) "Option": defines a selectable variability of the framework and (*VI*) "OptionGroup": group selectable variabilities of the framework. The last two elements are used to represent the variabilities provided by the CF that may be chosen by the application engineer. The "Group" element is also employed to support feature hierarchy, however, details on feature selection are not shown on this paper. Nevertheless, more details related to both feature selection and the tool can be seen at Durelli et al's paper [**?**]. This tool provides fully computational environment to the approach herein described.

Both of our proposed models have identical appearance, however, they are employed in different moments. The first proposed model, the RRM, is a graphical documentation regarding the Reuse Requirements, which are related to the information needed to couple the CF to a base application, which is conventionally part of "cookbook". This model contains all of information regarding all CF features and should be provided by a framework engineer. The second model, the RM, is a subset of the RRM that only contains the features selected for reuse. 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 that contain boxes, as seen in Figure 2. Each box is an instance of a concrete metaclass element and represent a reuse requirement. They also contain three lines, the first line contains a icon of the element type, which is the same type visible in the "Palette", and a name for the reuse requirement. The second line shows a description to facilitate the comprehension of the application engineer and the last line is filled by the application engineer to provide the information regarding the base application. Note that the last line is only used in RMs.

By analyzing the RRM, an application engineer should be able to learn which informations are required by the framework during the reuse process. This model also represents the variabilities provided by a framework that must be chosen by an application engineer. In order to instantiate a framework, the RRM may indicate the need of informing join-points of the base code where crosscutting behavior would be applied to, as well as classes, interfaces or aspect names that would be affected. Framework variabilities that must be chosen during reuse process are also visible. For example, to be able to instantiate a persistence CF, the application engineer must specify methods from base application that should be executed after a database connection is opened and before it is closed. It is also needed to specify methods that represent data base transactions, and the variabilities must be chosen, e.g., the driver which should be used to connect to the database system.

The other model, the RM, is shown in Figure 2. It supports the reuse process of a crosscutting framework. It is intended that the reuse process can be completely executed by completing the third line of the boxes of this model. Therefore, it should be used by the application engineer in order to reuse a framework. For instance, the value "*base.Customer.opening()*" is a method of the base application and was inserted by the application engineer in the third line of box "Connection Opening".

After the application engineer fills in the RM with the information needed by an member of a CF, it is possible to generate the final reuse code. To illustrate the use of these models we have used the "persistence" CF described in Section II.

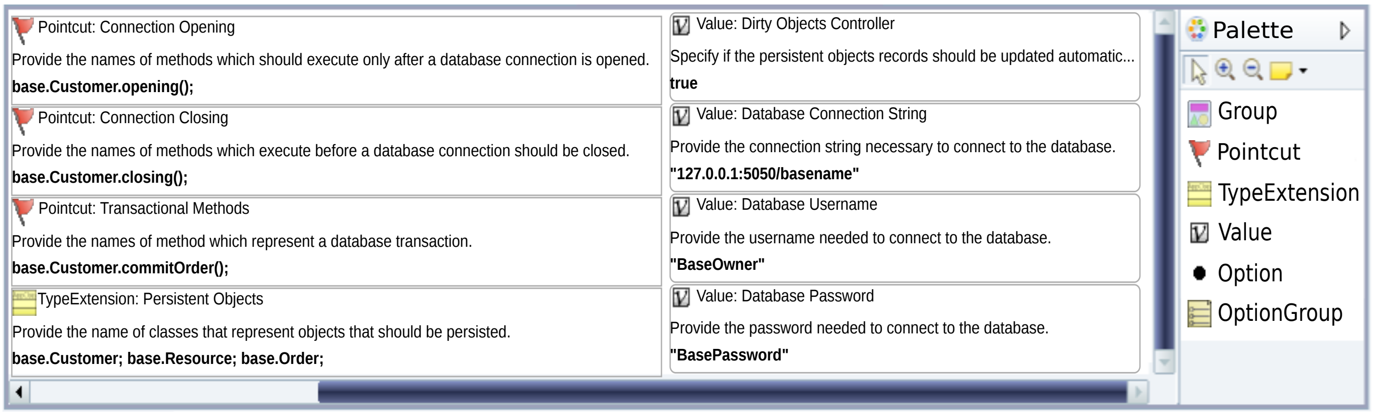
A. Reuse Example

In this section, we briefly show an example of how to use our models and generate code in order to reuse a CF. To reuse the "persistence" CF, the application engineer must specify explicitly which features will be used in the base application. This is important because usually the CFs have a great deal of features that probably will not be used in the application base. Therefore, we developed an environment to facilitate the feature selection in order to instantiate a member. This environment also provides a way to validate the combinations of features. Further details of this environment are out of the scope of this paper.

In Figure 2, there is our model editor. By using this editor, it is possible to model RRMs and RMs. In this example there is a RM related to the "persistence" CF being completed with information of a base application. The pointcuts "Connection Opening", "Connection Closing" and "Transaction Methods" are intended to capture specific join-points of the base application, e.g. names of methods of the base application that will be affected by the framework. The first two represent, respectively, method executions that should occur after a database connection is open or before it is closed. The last pointcut represents methods that encapsulate data transactions.

The "Persistent Objects" is a type extension definition, then, it may represent either a class or an interface that should be extended or implemented by a base class or interface. In this case, the application engineer must supply names of classes (or their super-types) which represent objects that should be persisted on the database.

"Dirty Objects Controller" is a boolean value which is used to define if the dirty objects controller should be active. This is used to update the database records automatically as soon as


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"

Fig. 2. Reuse Requirements Models and Reuse Models editor

any attribute belonging to a persistent object is changed by a set method. “Database Username” and “Database Password” are string values that are used to define the username and password needed to log into the database system. “Database Connection String” is a string value which should be used to specify the database connection details, i.e., the database system address, port and database name.

After completing the Reuse Model, it is possible to execute a code generator, which is a model to code transformation tool capable of generating reuse code in AspectJ, illustrated on Figure 3, which allows coupling the base application to the framework in a separate module. The final software is the composition of base application code, reuse code for each reused framework and the code of reused frameworks.

```
public aspect ConnectionCompositionReuse
            extends ConnectionComposition {
            public pointcut openConnection():
                execution  ( * base.Customer.opening());
            public pointcut closeConnection():
                execution  ( * base.Customer.closing());
            public pointcut transactional():
                execution  ( * base.Customer.commitOrder());
}
public aspect OORelationalMappingReuse
            extends OORelationalMapping {
            declare parents: base.Customer
                        implements PersistentRoot;
            declare parents: base.Resource
                        implements PersistentRoot;
            declare parents: base.Order
                        implements PersistentRoot;
}
public aspect ConnectionValues {
            public String SelectedManager.setDSN() {
                        return "127.0.0.1:/basename";
            }
            public String SelectedManager.setUsername() {
                        return "BaseOwner";
            }
            public String SelectedManager.setPassword() {
                        return "BasePassword";
            }
}
```

Fig. 3. Reuse Code Fragment

The first code of Figure 3 contains a new aspect which was created by generating code for the three pointcuts of the RM. This aspect extends an abstract aspect of the framework with the supplied information. In the second code, the type extension is implemented, then the classes written in the RM, “Customer”, “Resource” and “Order”, receive an interface of the framework, which is used to apply crosscutting behavior. In the third code, the value definitions are set by overriding methods of the framework.

IV. Evaluation

Two experiments were conducted to compare our reuse tool with the conventional technique. The first experiment is called “Reuse Study” and was planned to identify the gains in productivity when reusing a framework, which is the main objective of our tool. The second experiment is referred as “Maintenance Study” and was planned to identify whether the reuse models help or not the maintenance of an application that uses a CF and needs modifications. This second study is important because maintenance activities are usually performed more times than the reuse process. Each experiment was applied twice. In this paper, the first execution is referred as “Primary” and the second execution is referred as “Secondary”. Since there are two executions for each experiment, we present four study executions in this section.

A. Reuse Study Definition

The objective is to compare the effort of reusing frameworks by using a conventional technique with by using a model-based technique. The Persistence CF briefly presented in Section II played the role of “study subject” and it was used in both reuse techniques (conventional and model-based). The quantitative focus was determined considering the time spent in conducting the reuse process and the qualitative focus was to determine which technique takes less effort during reuse process. This experiment was conducted from the perspective of application engineers reusing CFs and the study object is the ‘effort’ to perform a CF reuse.

B. Maintenance Study Definition

The objective was to compare the effort in modifying a CF-based application by editing the reuse code (conventional technique) with by editing the RM. The Persistence CF shown in Section II 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 takes less effort to edit during maintenance. This experiment was conducted from the perspective of application engineers who intend to maintain CF-based applications. The study object is the 'effort' to maintain a CF-based application.

C. Study Planning

The first experiment was planned considering the following question: "Which reuse technique takes less effort to reuse a CF?"; The second experiment was planned considering the question: "Which artifact takes less effort to edit during maintenance, reuse model or reuse code?"; We gathered and analyzed the timings taken to complete the process for each activity.

1) Context Selection: Both studies were conducted with students of Computer Science, in this section, they are referred as participants. Sixteen participants took part on the experiments, eight of these were undergraduate students and the other eight were post graduate students. Every participant had prior AspectJ experience.

2) Formulation of Hypotheses: The Table I contains our formulated hypotheses for the reuse study, which are used to compare the productivity of our tool and the conventional ad-hoc process. Both of these processes can be used to successfully reuse a CF and couple it to an application that has no reuse code.

TABLE I
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 - $Tm_r < 0$

There are two variables shown on the table: "Tc$_r$" and "Tm$_r$". "Tc$_r$" represents the overall time to reuse the framework using the conventional technique while "Tm$_r$" represents the overall time to reuse the framework using the model-based tool. There are three hypotheses shown on the table: "H0$_r$", "Hp$_r$" and "Hn$_r$". The "H0$_r$" hypothesis is true when both techniques are equivalent; then, the time spent using the conventional technique minus the time spent using the model-based tool is approximately zero. The "Hp$_r$" hypothesis 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. The "Hn$_r$" hypothesis 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 exactly one of them is true.

The formulated hypotheses for the maintenance study are listed on Table II. These hypotheses consider the outcome of comparing the edition of the reuse code (conventional technique) with our approach (model-based).

TABLE II
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 - $Tm_m < 0$

The Table II also contains three variables. "Tc$_m$" represents the overall time to edit a reuse code during maintenance while "Tm$_m$" represents the overall time to edit the reuse model during maintenance. The "H0$_m$" hypothesis is true when the edition of both artifacts is equivalent. The "Hp$_m$" hypothesis is true when the edition of the reuse code takes longer than editing the RM. The "Hn$_m$" hypothesis is true when the edition of the reuse code takes less time than editing the RM. These hypotheses are also mutually exclusive and exactly one of them is true.

3) Variable Selection: The dependent variables are those which we analyze in this work. For each study, we provide analysis of the "time spent to complete the process". The independent variables are controlled and manipulated, for example, "Base Application", "Technique" and "Execution Types".

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.

5) Design of the studies: The participants were divided into two groups. Each group was composed by four post graduate students and four undergraduate students. Each group was also balanced considering a characterization form and their results from the pilot study. On Table III, there are the phases planned for both studies.

6) Instrumentation for the Reuse Study: Base applications were provided along with two documents. The first document is a manual regarding the current reuse technique, and the second document is a list of details, which describes the

TABLE III
STUDY DESIGN

Phase	Group 1	Group 2
General Training	Reuse and Maintenance Training	
	Repair Shop	
1st Reuse Pilot Phase	Conventional	Models
	Hotel Application	
2nd Reuse Pilot Phase	Models	Conventional
	Library Application	
1st Primary Reuse Phase	Conventional	Models
	Deliveries Application	
2nd Primary Reuse Phase	Models	Conventional
	Flights Application	
1st Secondary Reuse Phase	Conventional	Models
	Medical Clinic Application	
2nd Secondary Reuse Phase	Models	Conventional
	Restaurant Application	
1st Primary Maintenance Phase	Conventional	Models
	Deliveries Application	
2nd Primary Maintenance Phase	Models	Conventional
	Flights Application	
1st Secondary Maintenance Phase	Conventional	Models
	Medical Clinic Application	
2nd Secondary Maintenance Phase	Models	Conventional
	Restaurant Application	

classes, methods and values regarding the application to be coupled which are needed when reusing the framework.

The applications provided had the same reuse complexity, then, in order to reuse each application, the participants had to specify four values, twelve methods and six classes. Each phase row of the Table III is divided into the name of the application and the technique employed to reuse the framework. For instance, during the 1st Primary Reuse Phase, the participants of the first group coupled the framework to the "Deliveries Application" by using the conventional technique, while the participants of the second used the model-based tool to perform the same exercise.

7) Instrumentation for the Maintenance Study: The base applications provided for the second study were modified versions of the same applications supplied during the first study. These applications were provided with incorrect reuse codes (conventional) and reuse models (model-based), which should be fixed by the participants. The participants received a manual regarding generic errors that could happen when the reuse code or model is incorrectly defined. It is important to point that the manual did not have details regarding the base applications, then, the participants had to find the errors by themselves by browsing the source code.

The applications provided had the same reuse complexity and the reuse codes and models had the same amount of errors. Then, 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 point that errors specific to manual edition of reuse code were not inserted in this study. The phases are also listed on Table III. That table contains the name of the application and the technique employed during maintenance. For instance, during the 1st Primary Maintenance Phase, the participants of the first group had to fix the reuse code of the "Deliveries Application" , while the participants of the second had to fix the reuse model to perform the same exercise.

D. Operation

1) Preparation: At first, every student was introduced to the tool and was taught how to edit reuse codes and reuse models. During each phase of the reuse study, the students were required to reuse the CF with a provided application. During the maintenance study, the students had to fix a reuse code or reuse model to complete the process. Every participant had to reuse and maintain every application by using only one of the techniques in equal numbers. Also, at any moment of the experiment, each group was using a different technique than the other group.

2) Execution: Initially, the participants signed a consent form and then answered a characterization form. The characterization form had questions regarding knowledge about AspectJ constructs, Eclipse IDE and Crosscutting Frameworks.

After concluding the characterization forms, participants were trained on how to reuse the supplied CF with the model-based reuse tool and then conventionally. It is important to note that every participant already had a basic experience with AspectJ and the conventional reuse of crosscutting frameworks.

Following the training, the pilot experiment was executed. The participants were split into two groups considering the results of the characterization forms. The pilot experiment was intended to simulate the real experiments, except that the applications were different, but equivalent. During the pilot experiment, the participants were allowed to ask questions about any issues they did not understand during the training. This could affect the validity, then, the data from this activity was only used to rebalance the groups.

During the real experiments, the participants had to work with two applications starting with a different technique for each group. The secondary executions were replications of the primary executions with another two applications. They were created in order to avoid the risk of getting unbalanced results during the primary execution, since some data gathered during the pilot were rendered invalid.

3) Data Validation: The forms filled by the participants were confirmed with preliminary data gathered during the pilot study. The researchers also watched the information system notifications to confirm if the participants had concluded and the captured data.

4) Data Collection: The recorded timings during the reuse processes with both techniques are listed on the Table IV. The timings for the maintenance study are found on Table V. There are five columns in each of these tables, "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 post-graduate students and the rest are undergraduate students; "Time" column lists the time the participant spent to complete each phase.

The information system employed to gather the experiment data 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, then, the delay of transmission by the computers are not considered, which are believed to be insignificant in this case, because preliminary calculations considering the client clocks did not change the order of results.

That system was able to gather the timings and supplied information transparently. The participants only had to execute the start time, which was supervised, and work on the processes by themselves. Once the test case provided had 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.

TABLE IV
REUSE PROCESS TIMINGS

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

E. Data Analysis and Interpretation

The data of the first study is found on Table IV, which is ordered by the time taken to complete the process. The first notorious information found on this table is that the model-based reuse tool, which is identified by the letter ‘M’, is found on the first twelve results. The conventional process, which is identified by the letter ‘C’, got the last four results.

The timings data of Table IV is also represented graphically in a bar graph, which is plotted on Figure 4. The same identifying code for each participant and the elapsed time in seconds are visible on the graph. The bars for conventional technique and model tool use are paired for each participant, allowing easier visualization of the amount of time taken by each of them.

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

TABLE V
MAINTENANCE PROCESS TIMINGS

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

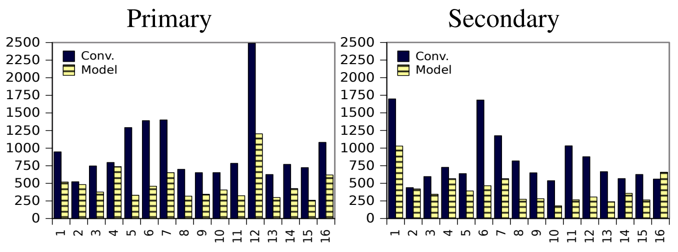


Fig. 4. Reuse Process Timings Bars Graph

The data of the second study is found on Table V. This study has provided similar results. The first eleven values were scored by using the model-based tool while the last four were scored by using the conventional tool. Only the participant number 16 was able to reuse the framework faster by using the conventional process, which contradicts the results taken from the same participant in the previous study. There is also a bar graph for this study in Figure 4.

The plots for the maintenance study are found on Figure 5, which also follow the same guidelines used while plotting the graphs for the previous study. Considering the timings of the maintenance study, the reuse model edition does not provide advantage in terms of productivity when maintaining an application that reuses a CF, since most of participants took longer to edit the model than the reuse code.

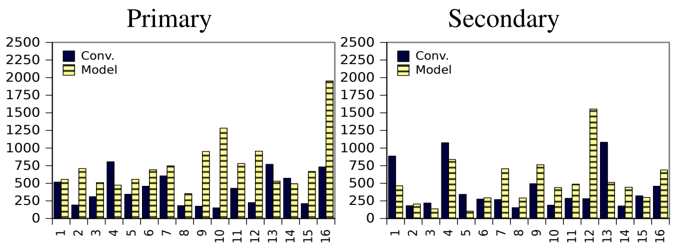


Fig. 5. Maintenance Process Timings Bars Graph

On Table VI there are average timings and their proportions. By considering the average time the participants of both groups needed to complete the processes, the conventional technique took approximately 97.64% longer than the model-based tool.

TABLE VI
AVERAGE TIMINGS

A.	Tech.	Avg.	Sum of Avg.	Percents
Reuse Study				
Primary	Conv.	16:13.44008	30:03.79341	66.7766%
Secondary		13:50.35333		
Primary	Model	08:04.980525	14:57.441176	33.2234%
Secondary		06:52.460651		
Total		45:01.234586		100.0000%
Maintenance Study				
Primary	Conv.	06:57.498758	13:55.762733	39.5521%
Secondary		06:58.263975		
Primary	Model	12:43.152626	21:17.305521	60.4479%
Secondary		08:34.152895		
Total		35:13.068254		100.0000%

F. *Hypotheses Testing*

In this section, we present statistical calculations to evaluate the data of both studies. We applied Paired T-Tests for each execution and another T-Test after removing eight outliers for each study. The seconds spent were processed using the statistic computation environment "R" [**?**]. Considering the reuse study, the results of the T-Tests are shown on Table VII. For the maintenance study, the same operation was executed and the results of the T-Test are shown on Table VIII.

The first columns of these tables contain the type of T-Test, the second columns indicate the source of the data, the "Means" columns indicate the resultant mean, which is the mean of the differences for an paired T-Test and one mean for each set for the other T-Test, which represent the conventional and the model-based tool means, respectively. The "d.t." columns stand for the degree of freedom; "t" and "p" are variables considered in the hypothesis testing.

The Paired T-Test is used to compare the the differences between two samples related to each participant, in this case, the time difference of every participant is considered individually, and 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 may be an outlier in a specific technique, which breaks the pairs. It is referred as two-sided because the two sets have the same number of elements, since the same number of outliers were removed from each group.

TABLE VII
REUSE STUDY T-TEST RESULTS

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

The "Chi-squared test" was applied on both studies in order to detect the outliers that were removed when calculating the unpaired T-Test, which is refered as "Two-sided". The results of the "Chi-squared test" for the reuse study are found on Table IX and the results of the same test for the maintenance study are found on Table X. 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 an comparison to the variance of the complete set and 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.

TABLE VIII
MAINTENANCE STUDY T-TEST RESULTS

T-Test	Data	Means	d.f.	t	p
Paired	Real	-345.6539	15	-3.971923	0.001227479
Paired	Spare	-95.88892	15	-1.191781	0.2518624
Two-Sided	Both	431.3323	24.22097	-2.662684	0.0135614
		641.0024			

TABLE IX
CHI-SQUARED TEST FOR OUTLIER DETECTION APPLIED ON REUSE STUDY

Study	T.	G.	X^2	p	position	outlier
Real	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
Spare	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

TABLE X
CHI-SQUARED TEST FOR OUTLIER DETECTION APPLIED ON MAINTENANCE STUDY

Study	T.	G.	X^2	p	position	outlier
Real	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
Spare	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

In order to achieve better visualization of the outliers, we also provide two plots of the data sets. In Figures 6 and 7 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.

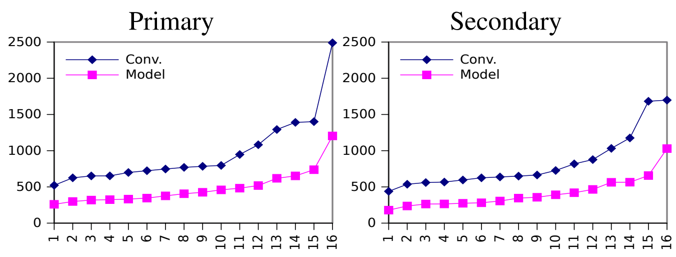


Fig. 6. Reuse Process Timings Bars Graph

Considering the reuse study and according to the analysis from Table VII, since all p-values are less than the margin

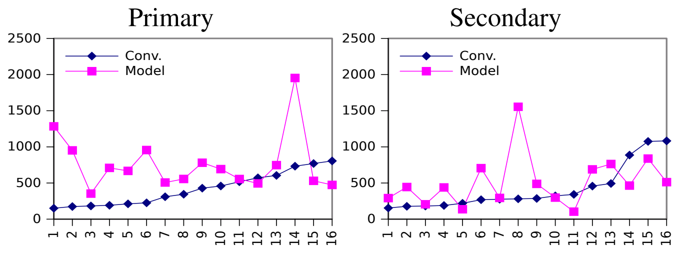


Fig. 7. Maintenance Process Timings Bars Graph

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 considers that the conventional technique takes more time than our tool.

Considering the maintenance study and according to the analysis from Table VIII, since all p-values are bigger than the margin of error (0.01%), which corresponds to the established significance level of 99.99%, then, statistically, we cannot reject the "$H0_m$" hypothesis that states the techniques are equivalent. Therefore, statistically, we can assume that the effort needed to edit a reuse code and a reuse model is approximately equal.

G. Threats to Validity

Internal Validity:

- Experience Level of Participants: the varied participant knowledge that could affect the collected data. To mitigate this threat, we divided the participants in two balanced groups considering the experience level and rebalanced the groups considering the preliminary results. Also, the participants had prior experience on how to reuse the CF conventionally. During the training, the participants were trained on how to reuse the CF with the model-based tool and then again on how to reuse it conventionally, which could cause the participants to have more experience with the conventional technique.
- Productivity under evaluation: there is a possibility that this might influence the experiment results because students often tend to think they are being evaluated by experiment results. 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 installations could affect the recorded timings. However, the different groups used the same configuration, make, model and operating system in equal numbers and the participants were not allowed to change their machines during in the same activity, which means that a participant could not reuse a framework conventionally by using a different computer that was used to reuse it with our tool.

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.

External Validity:

- Interaction between configuration and treatment: it is possible that the reuse exercises are not accurate for every reuse of a crosscutting framework for real world applications. Only a single crosscutting framework was considered and the base applications had the same complexity. To mitigate this threat, the exercises were designed considering applications based on the real world.

Conclusion Validity:

- Measure reliability: it refers to metrics used to measuring the reuse effort. To mitigate this threat we have used only the time taken which was captured by an information system in order to allow greater precision;
- Low statistic power: the ability of a statistic test in reveal reliable data. To mitigate we applied three T-Tests to statistically analyze the experiment data.

V. Related Work

The approach proposed by Cechticky *et al.* [**?**] allows object-oriented application framework reuse by using a tool called OBS Instantiation Environment. That tool supports graphical models do define the settings of the expected application to be generated. The model to code transformation generates a new application that reuses the framework.

The proposal found in this paper differs from their approach on the following topics: 1) their approach is restricted to frameworks known during the development of the tool; 2) it does not use aspect-orientation; 3) the reuse process is applied on application frameworks, which are used to create new applications.

Another approach was proposed by Oliveira *et al.* [**?**]. Their approach can be applied to a greater number of object oriented frameworks. After the framework development, the framework developer may use the approach to ease the reuse by writing the cookbook in a formal language known as Reuse Definition Language (RDL) which also can be used to generate the source code. This process allows to select the variabilities and resources during reuse, as long as the framework engineer specifies the RDL code correctly.

These approaches were created to support the reuse during the final development stages. Therefore, the approach proposed in this paper differs from others by the 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.* [**?**] is specific for only one framework, its can be employed since the design phase. The other related approach can be employed in a higher number of frameworks, however it is used in a lower abstraction level, and does not support the design phase. Other difference is the generation of aspect-oriented code, which improves code modularization.

VI. Conclusions

In this paper, a model-based process was presented, which raises abstraction levels of CF reuse. It serves as a graphical view that replaces textual cookbooks and is used to perform the reuse in a model driven approach. From our proposed model-based approach, a new reuse process was delineated, which employs the forms during the development of a new application, allowing engineers to start the reuse since earlier software development phases and reduce the time to reuse a CF. With this, application developers do not need to worry about reuse coding issues nor how the framework was implemented, allowing to focus on the reuse requirements in a higher abstraction level.

Our approach was evaluated in two experiments that could answer the questions of the study planning, which indicate their conclusive success. The links for the gathered data can be accessed on *http://www2.dc.ufscar.br/˜valter/*. The results regarding the productivity of reuse process were promising. However, the results of the maintenance study showed that our technique has no disadvantages in maintenance effort.

Furthermore, we have identified some limitations related to our research project. Once the models have been devised on top of the Eclipse Modeling Project, they can not be used in another environment. Furthermore, the code generator only generates code for Java and AspectJ, therefore, only frameworks developed in these languages are currently supported.

It is also important to point that our tool is part of a project to develop an integrated development environment for CF, which currently supports CF feature subset selection and a CF repository service. It is important to note that our tool also supports CFs that do not employ feature selection, in these cases, the RRM and RMs would be exactly equal.

However, we have not yet evaluated how to deal with coupling multiple CFs to a single base application. Despite this functionality already being supported, some frameworks may conflict with each other and lead to unwanted results.

The code generated is based on AspectJ and it was not evaluated if it supports every CF without modifications. Although not stated, we have also worked on selecting subsets of features of the framework.

Long term future works regard: (*i*) carry out a experiment using other CF, for verifying if the models proposed assist both the reuse and to maintain a reuse code; (*ii*) execute a experiment to verify whether the abstraction of the elements related to the models are sufficiently ideal; (*iii*) evaluate the standpoint of the domain engineers/frameworks, (*iv*) improve the elements of the models, i.e., better them graphically and (*v*) analyze the reusability of the abstract metamodel's metaclasses.

Acknowledgments

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 paper was created. Thiago Gottardi would also like to thank FAPESP (Process 2011/04064-8).

References

[1] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *2007 Future of Software Engineering*, ser. FOSE 07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 37–54.

[2] AspectJ Team, "The AspectJ(tm) programming guide," 2003.

[3] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, 3rd ed. Addison-Wesley Professional, 2001.

[4] M. Fayad and D. C. Schmidt, "Object-oriented application frameworks," *Commun. ACM*, vol. 40, pp. 32–38, October 1997. [Online]. Available: http://doi.acm.org/10.1145/262793.262798

[5] M. Mortensen and S. Ghosh, "Creating pluggable and reusable non-functional aspects in AspectC++," in *Proceedings of the Fifth AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software*, 2006.

[6] V. Shah and V. Hill, "An aspect-oriented security framework: Lessons learned," in *Proceedings of AOSDSEC'04 (AOSD Technology for Application-Level Security). Workshop of the Aspect Oriented Software Development Conference*, Lancaster, UK, March, 23 2004.

[7] S. Soares, E. Laureano, and P. Borba, "Distribution and persistence as aspects," *Software: Practice and Experience*, vol. 33, no. 7, pp. 711–759, 2006.

[8] U. Kulesza, E. Alves, R. Garcia, C. J. P. D. Lucena, and P. Borba, "Improving extensibility of object-oriented frameworks with aspect-oriented programming," in *Proc. of the 9th Intl Conf. on Software Reuse (ICSR'06)*, 2006, pp. 231–245.

[9] V. Camargo and P. Masiero, "Frameworks orientados a aspectos," in *Anais Do 19^o Simpósio Brasileiro De Engenharia De Software (SBES'2005), Uberlândia-MG, Brasil, Outubro.*, 2005.

[10] M. Huang, C. Wang, and L. Zhang, "Towards a reusable and generic aspect library," in *Workshop of the Aspect Oriented Software Development Conference at AOSDSEC'04*, Lancaster, UK, March, 23 2004.

[11] I. Zanon, V. V. Camargo, and R. A. D. Penteado, "Reestructuring an application framework with a persistence crosscutting framework," *INFOCOMP*, vol. 1, pp. 9–16, 2010.

[12] R. Lazanha, A. Oliveira, R. Penteado, R. Ramos, O. Pastor, and V. Camargo, "Uma arquitetura de referência baseada em papéis para frameworks transversais de persistência: Uma análise quantitativa," in *XXXVI Clei – Conferência Latino-Americana de Informática*, Assunção, Paraguay, 2010.

[13] M. Bynens, D. Landuyt, E. Truyen, and W. Joosen, "Towards reusable aspects: The mismatch problem," in *Workshop on Aspect, Components and Patterns for Infrastructure Software (ACP4IS'10)*, 2010, pp. 17–20.

[14] D. Sakenou, K. Mehner, S. Herrmann, and H. Sudhof, "Patterns for re-usable aspects in object teams," in *Net Object Days*, Erfurt, 2006.

[15] C. Cunha, J. Sobral, and M. Monteiro, "Reusable aspect-oriented implementations of concurrency patterns and mechanisms," in *Aspect-Oriented Software Development Conference (AOSD'06)*, Bonn, Germany, 2006.

[16] N. Soudarajan and R. Khatchadourian, "Specifying reusable aspects," in *Asian Workshop on Aspect-Oriented and Modular Software Development (AOAsia'09)*, 2009.

[17] V. V. de Camargo and P. C. Masiero, "An approach to design crosscutting framework families," in *Proceedings of the 2008 AOSD workshop on Aspects, components, and patterns for infrastructure software*. New York, NY, USA: ACM, 2008, pp. 3:1–3:6.

[18] Eclipse Consortium, G*raphical* M*odeling* F*ramework, version 1.5.0*, http://www.eclipse.org/modeling/gmp/, Graphical Modeling Project Std., 2011.

[19] Free Software Foundation, Inc. , "R," http://www.r-project.org/, December 2011.

[20] V. Cechticky, P. Chevalley, A. Pasetti, and W. Schaufelberger, "A generative approach to framework instantiation," in *Proceedings of the 2nd international conference on Generative programming and component engineering*, ser. GPCE '03. New York, NY, USA: Springer-Verlag New York, Inc., 2003, pp. 267–286. [Online]. Available: http://portal.acm.org/citation.cfm?id=954186.954203

[21] T. C. Oliveira, P. Alencar, and D. Cowan, “Reusetool-an extensible tool support for object-oriented framework reuse,” *J. Syst. Softw.*, vol. 84, no. 12, pp. 2234–2252, Dec. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2011.06.030