SPL-TuPI: A Tool to support Product Instantiation of Software Product Line Projects

Pablo F. Matos[1,2], Djan A. Santos[1,2], Crescencio R. L. Neto[1,2], Eduardo S. Almeida[2]

[1]Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA)
Av. Amazonas 3150 – Zabelê – Vitória da Conquista – BA – Brasil

[2]Universidade Federal da Bahia (UFBA)
Av. Adhemar de Barros, s/n, Campus de Ondina – Salvador – BA – Brasil

{pablofmatos, djan.santos, crescenciolima}@ifba.edu.br, esa@dcc.ufba.br

Abstract. *Software product line (SPL) is a set of products developed from a feature model satisfying the specific needs of a particular domain. The challenge of using the feature model is to derive product variability. The Java programming language does not natively have conditional compilation directives like C language, requiring the use of a preprocessor to generate product variability. So, it is not easy to create the product instantiation with the preprocessor. To address this problem, this paper presents a tool that implements a feature-oriented approach, called SPL-TuPI, to support product instantiation of SPL projects. We also report an experience on using the proposed tool with the product instantiation from a SPL application in the event management domain.*

Video link: *https://youtu.be/yadPJ1E_Ymo*

1. Introduction

The development of larger and more complex software systems demands better support for reusable software artifacts [1]. Software Product Line (SPL) is an important approach to address these demands. For this reason, SPL has been increasingly adopted in software industry [2, 3]. SPL is a set of software systems that share a common set of features satisfying the specific needs of a domain [1]. Besides language-based techniques (*e.g.*, Components and Services, Parameters, Design Patterns, Frameworks), which encode variability with available concepts within programming languages, external tools may be used to implement and manage variability [3].

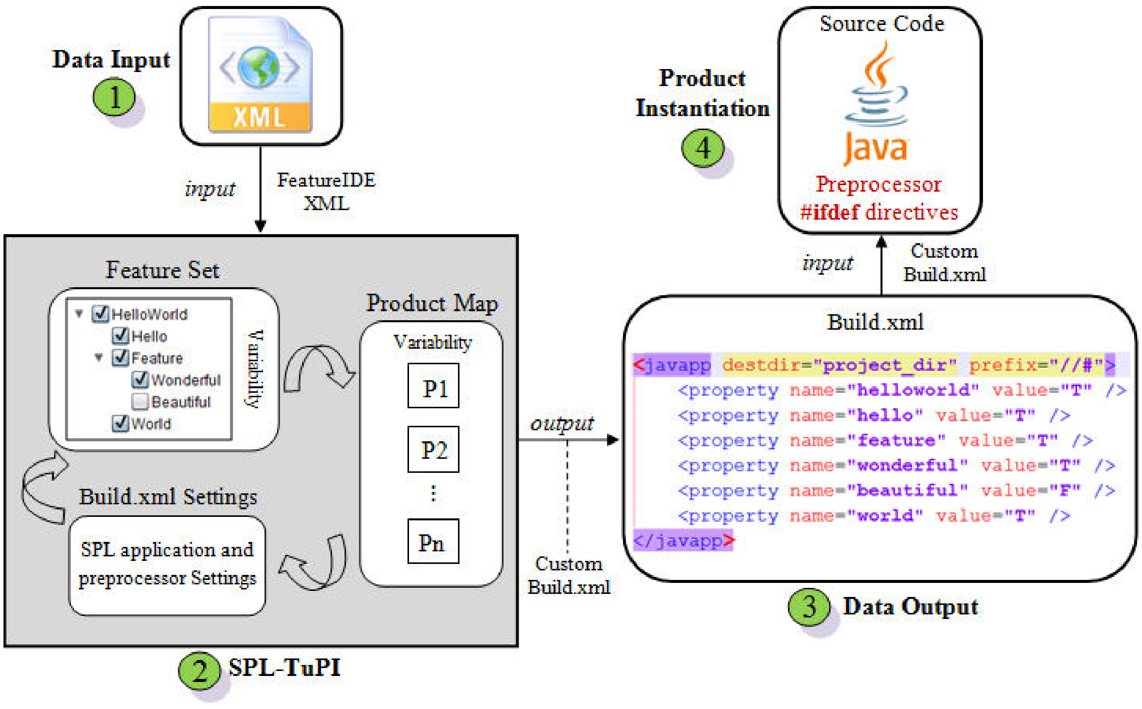
According to [1], the process of enabling and disabling features in a feature model for a new software product configuration is crucial to SPL implementation. In other words, a feature-oriented approach makes features explicit in requirements, design, code, and testing across the entire product line life cycle [3]. Regarding the development phase, a preprocessor is required to enable the variability implementation. However, there are some programming languages (*e.g.*, Java) that did not provide conditional compilation directives.

Moreover, during the development phase, SPL developers should map the products and synchronize them with the features. This challenge is combined with another demanding task, which is the variability management. Therefore, it is necessary to have tool support to aid the developers during the product instantiation. Despite the number of tools available [4], they still lack important tasks such as products source code and document generation.

In this context, this paper presents a tool, called *SPL-TuPI*, to support product instantiation of SPL projects. The main goal of the tool is to allow the user to configure a feature set dynamically to derive product variability for software product lines application. The remainder of this paper is organized as follows: Section 2 presents an overview of the *SPL-TuPI* tool; Section 3 presents a case study to evaluate the proposed tool; Section 4 presents related work; and Section 5 concludes this work and addresses future work.

2. *SPL-TuPI* Tool

This section presents the contributions that *SPL-TuPI*[1] provides to support product instantiation of SPL projects. This tool implements a feature-oriented approach to assist users during the features configuration process, in order to make the product instantiation simple.



```
<javapp destdir="project_dir" prefix="//#">
    <property name="helloworld" value="T" />
    <property name="hello" value="T" />
    <property name="feature" value="T" />
    <property name="wonderful" value="T" />
    <property name="beautiful" value="F" />
    <property name="world" value="T" />
</javapp>
```

Figure 1. Overview of the *SPL-TuPI*

Figure 1 shows the overview of the tool that is composed by four steps:

1. Data Input (*Step 1*), the feature set may be automatically imported from the XML file that is generated by the FeatureIDE tool [5]. Besides the import functionality, the user may dynamically create and delete features by itself;
2. *SPL-TuPI* tool (*Step 2*), the user may select which features will be part of the product instantiation. There is the possibility to define the product variability, creating the product to be included in the product map. It is also necessary to inform the location of the SPL application and the preprocessor. For each software product line, the user *may* select the feature set and define the product map, and he *must* configure the SPL application and the preprocessor;

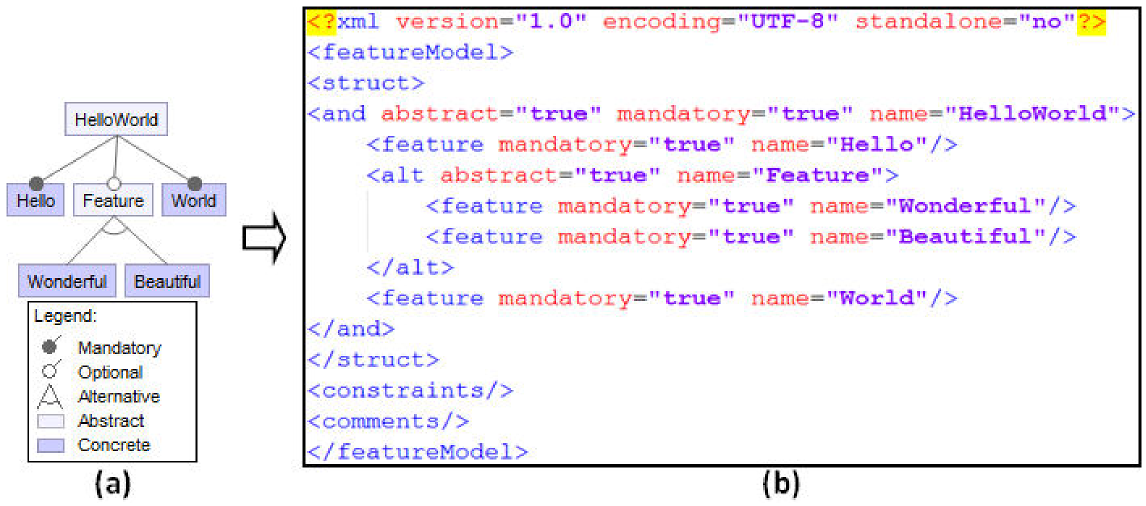
[1] http://bsi.conquista.ifba.edu.br/~pablofmatos/tool/spl-tupi

3. Data Output (*Step 3*), the tool's output is a customized file named as build.xml, which it contains the features set selected by the user only for one product instantiation at a time, and the information of the SPL application and the preprocessor, previously defined in Step 2; and
4. Product Instantiation (*Step 4*), the output file is used to instantiate the product of an SPL application by the preprocessor, through *#ifdef* directives inserted in the source code.

2.1. Data Input (Step 1)

Data input may be in two ways: (1) from the creation of the own features set by the user with the functionalities of addition and removal of features dynamically; (2) and from importing the XML file generated by the FeatureIDE tool [5]. FeatureIDE is an Eclipse plug-in that supports the feature-oriented software development. The first way will be shown in the next section (Step 2).

As may be seen in the feature model presented in Figure 2-a), the feature may be abstract or concrete. A feature is abstract if it is not mapped to implementation artifacts and concrete otherwise [5]. Connections between a feature and its group of subfeatures are distinguished as *and*-, *or*-, and *alternative-groups* [6]. If a feature is selected, all mandatory subfeatures of an *and-group* must be selected. In *or-groups*, at least one subfeature must be selected and in *alternative-groups*, exactly one subfeature has to be selected [5]. The feature may also be optional or mandatory, and the selection of a feature implies the selection of its parent feature.



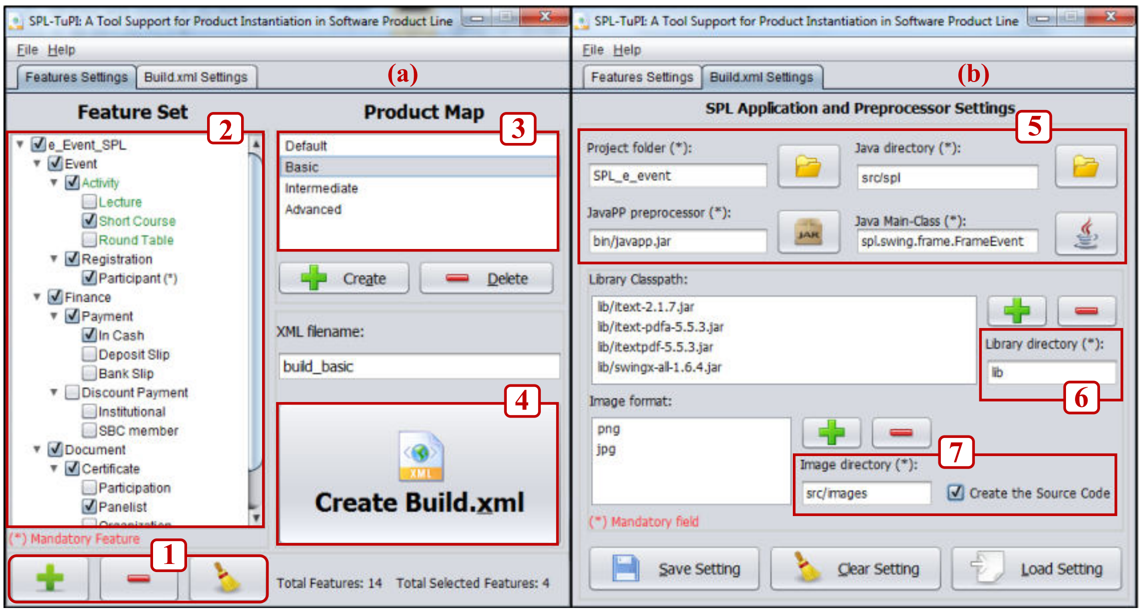
```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<featureModel>
<struct>
<and abstract="true" mandatory="true" name="HelloWorld">
    <feature mandatory="true" name="Hello"/>
    <alt abstract="true" name="Feature">
        <feature mandatory="true" name="Wonderful"/>
        <feature mandatory="true" name="Beautiful"/>
    </alt>
    <feature mandatory="true" name="World"/>
</and>
</struct>
<constraints/>
<comments/>
</featureModel>
```

Figure 2. Data input: (a) Feature Model (FM), (b) XML generated from the FM

Figure 2-b) shows an example of the XML file generated from the feature model by the FeatureIDE tool. The feature set is organized under the "*struct*" tag. This tag allows identifying the features and subfeatures, if they are abstract or concrete, if they are optional or mandatory, and their respective groups.

2.2. Main Functionalities (Step 2)

Figure 3 shows the *SPL-TuPI* tool screenshot with two tabs: The first tab deals with feature settings (Figure 3-a) and the second one deals with build.xml settings (Figure 3-b). The user may create the own feature set or delete the feature, respectively, by the functionalities of addition and removal of feature (item 1). The feature set is loaded in the TreeView component (item 2), in order to facilitate the hierarchical view and the feature selection by the user. The user may define many product lines as he wants (item 3). Then, there are two ways to set the products: The first one (item 2) presents the selection of features individually; the second one (item 3) presents the feature selection based on the product map. Finally, item 4 shows the button to activate the product build, in which it generates the XML file (*i.e.*, build.xml) for only one product instantiation at a time.



```
☑ e_Event_SPL
  ☑ Event
    ☑ Activity
      ☐ Lecture
      ☑ Short Course
      ☐ Round Table
    ☑ Registration
      ☑ Participant (*)
  ☑ Finance
    ☑ Payment
      ☑ In Cash
      ☐ Deposit Slip
      ☐ Bank Slip
    ☐ Discount Payment
      ☐ Institutional
      ☐ SBC member
  ☑ Document
    ☑ Certificate
      ☐ Participation
      ☑ Panelist
(*) Mandatory Feature
```

Product Map: Default, Basic, Intermediate, Advanced

XML filename: build_basic

Total Features: 14 Total Selected Features: 4

Project folder (*): SPL_e_event
Java directory (*): src/spl
JavaPP preprocessor (*): bin/javapp.jar
Java Main-Class (*): spl.swing.frame.FrameEvent
Library Classpath: lib/itext-2.1.7.jar, lib/itext-pdfa-5.5.3.jar, lib/itextpdf-5.5.3.jar, lib/swingx-all-1.6.4.jar
Library directory (*): lib
Image format: png, jpg
Image directory (*): src/images
☑ Create the Source Code
(*) Mandatory field

Figure 3. *SPL-TuPI* tool: (a) Features settings, (b) Build.xml settings

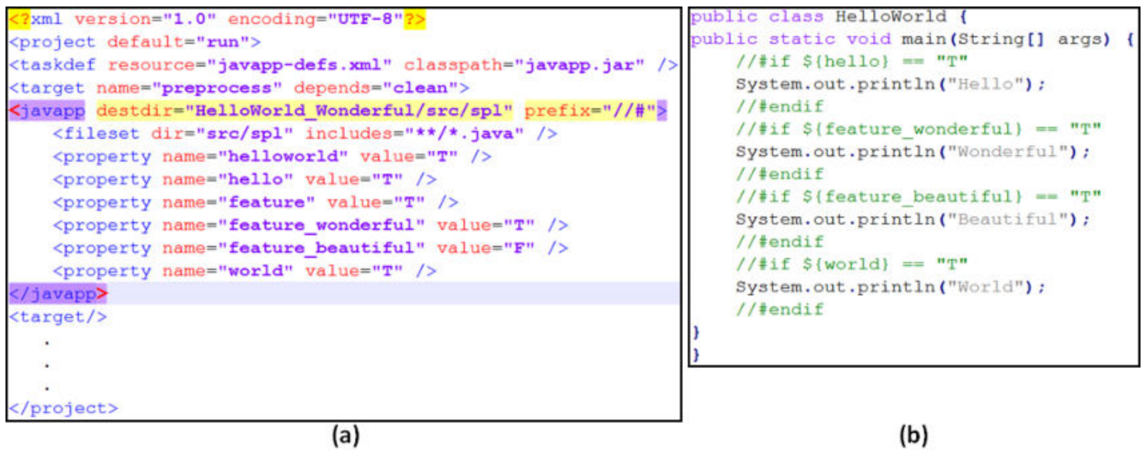
Before creating the XML file indeed, it is mandatory to set the location of the SPL application and the preprocessor (Figure 3-b). First, the folder of the SPL application must be chosen (item 5). Since the Java programming language did not support conditional compilation, we use the open source JavaPP [7] to perform the source code preprocessing. The JavaPP is a Java preprocessor to automatically run Apache Ant[2] files. Then, it is necessary to set the JavaPP preprocessor file (item 5). The directory that contains all Java classes and the project's main class must also be defined (item 5). Other mandatory settings from the SPL application are: the libraries directory used in the project (item 6), and the image directory also used in the project (item 7). Along with the application, the user may also make available the source code. In this case, the user must click on the checkbox "*Create the Source Code*" (item 7).

2.3. Data Output (Step 3) and Product Instantiation (Step 4)

Figure 4-a) shows the output file, named as build.xml, ready to be used by the SPL application. Besides the information of the SPL application, the XML file also contains the information of the JavaPP preprocessor, in which it will be interpreted by Apache Ant. The JavaPP syntax is the same as the C preprocessor [8]. The goal of this file is to

[2] Apache Ant (https://ant.apache.org/manual/) is a tool for automating software build process.

make possible the product instantiation. Figure 4-b) shows the selected features incorporated in the SPL application through *#ifdef* directives inserted in source code.



```
<?xml version="1.0" encoding="UTF-8"?>
<project default="run">
<taskdef resource="javapp-defs.xml" classpath="javapp.jar" />
<target name="preprocess" depends="clean">
<javapp destdir="HelloWorld_Wonderful/src/spl" prefix="//#">
    <fileset dir="src/spl" includes="**/*.java" />
    <property name="helloworld" value="T" />
    <property name="hello" value="T" />
    <property name="feature" value="T" />
    <property name="feature_wonderful" value="T" />
    <property name="feature_beautiful" value="F" />
    <property name="world" value="T" />
</javapp>
<target/>
    .
    .
    .
</project>
```

(a)

```
public class HelloWorld {
public static void main(String[] args) {
    //#if ${hello} == "T"
    System.out.println("Hello");
    //#endif
    //#if ${feature_wonderful} == "T"
    System.out.println("Wonderful");
    //#endif
    //#if ${feature_beautiful} == "T"
    System.out.println("Beautiful");
    //#endif
    //#if ${world} == "T"
    System.out.println("World");
    //#endif
}
}
```

(b)

Figure 4. (a) Output file, (b) SPL Application source code with #ifdef directives

2.4. Implementation

The source code was developed through the J2EE platform with the Java Swing component and the external library, called JDOM[3] (version 2.0.6), was used to manipulate the XML files.

3. Case Study: e-Event SPL

We use the *SPL-TuPI* tool to instantiate the e-Event SPL that was implemented using the Java language. The case study's domain chosen for the SPL project consists in the event management domain. The e-Event SPL platform was conceived based on largely used conference management systems, such as: EasyChair[4], JEMS[5], and CyberChair[6].

Figure 5 shows the e-Event SPL feature model which has 22 features, 3 being abstract features (Event, Finance and Document) and 19 being concrete features (all other features may be seen in Figure 5).

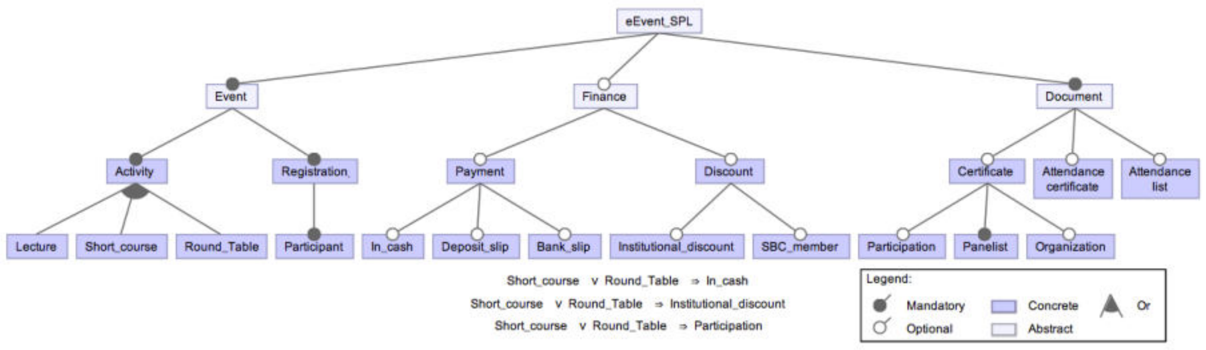


Figure 5. Feature Model from e-Event SPL

[3] http://www.jdom.org/
[4] http://www.easychair.org/
[5] https://submissoes.sbc.org.br/
[6] http://www.borbala.com/cyberchair/

The e-Event SPL consists of three pre-configured products: basic, intermediate, and advanced versions. In each one of them, it was defined the feature set that meet the demands of different types of conferences. Table 1 presents the e-Event SPL product map after the investigation of the conference management systems.

Table 1. Product Map from e-Event SPL

Feature	*Subfeature*	*Basic e-Event*	*Intermediate e-Event*	*Advanced e-Event*
Activity	Lecture	No	No	**Yes**
	Short Course	**Yes**	**Yes**	**Yes**
	Round Table	No	**Yes**	**Yes**
Registration	Participant	**Yes**	**Yes**	**Yes**
Payment	In Cash	**Yes**	**Yes**	**Yes**
	Deposit Slip	No	**Yes**	**Yes**
	Bank Slip	No	No	**Yes**
Discount	Institutional Discount	No	**Yes**	**Yes**
	SBC Member	No	No	**Yes**
Certificate	Participation	**Yes**	**Yes**	**Yes**
	Panelist	No	**Yes**	**Yes**
	Organization	No	No	**Yes**
Document	Attendance Certificate	No	No	**Yes**
	Attendance List	No	**Yes**	**Yes**

3.1. Metrics Analysis

Table 2 shows the product configurations and LOC metric[7] about the e-Event SPL, in which the product line is organized in four categories, namely: (1) product without any feature; (2) basic product; (3) intermediate product; and (4) advanced product, respectively with, 8, 51, 63 and 72 Java source files. The size of the source files without any library is, respectively, 460 KB, 690 KB, 768 KB and 811 KB. Comparing the size of the source file from the Category 2 and Category 3 is, respectively, 34.47% and 12.25% less code than the Category 4. It shows that unselected features are indeed not included in the product instantiation. For the record, the total Lines of Code (LOC) of the four product line is, respectively, 1.811, 7.723, 9.503 and 10.863.

Table 2. Product Configurations and LOC metric from e-Event SPL

Category	*Product Line*	*Product Configurations*			*LOC*
		Quantity of Source Files	*Size without Library*	*Size Comparison*	
1	Without Features	8	460 KB	---------	1.811
2	Basic	51	690 KB	230 KB (34.47%)	7.723
3	Intermediate	63	768 KB	308 KB (12.25%)	9.503
4	Advanced	72	811 KB	351 KB	10.863

3.2. Tool Instantiation

Figure 3 shows the proposed tool instantiated by the e–Event SPL. Figure 3-a) shows the feature set loaded and the three products created. Figure 3-b) shows the settings of the e-Event SPL application and the JavaPP preprocessor.

[7] http://www.locmetrics.com/

4. Related Work

FeatureIDE is an Eclipse-based framework to support feature oriented software development. The tool main focus is to cover the whole development process and to incorporate tools for the implementation of SPLs into an integrated development environment (IDE) [5]. FeatureIDE does not address the product instantiation with a preprocessor (*#ifdef*) in Java.

Machado *et al.* [9] present a tool, called SPLConfig, to support automatic product configuration in SPL which the main goal is to derive an optimized features set that satisfies the customer requirements, in order to maximize the benefit/cost without exceeding the available budget. The tool focuses only on the product configuration and does not focus on the product instantiation.

The GenArch [10] and its evolution GenArch+ [11], presented a model-based product derivation tool that combines the use of models and code annotations in order to enable the automatic product derivation of existing SPLs. However, these tools also do not address the product instantiation with preprocessor (*#ifdef*) in Java.

Recently, [4] presented a systematic literature review on SPL management tools. The study identified thirty-three tools that support the product derivation functionality.

5. Conclusion and Future Work

This paper presented the *SPL-TuPI*, a Java tool built to assist users during the feature configuration process in order to make the product instantiation simple. This tool implements a feature-oriented approach, in which it generates the product variability based on a XML file easily defined by the user. We also report an experience on using the proposed tool with the product instantiation from an SPL application in the event management domain, called e-Event SPL. The difficulty faced in the development of this SPL has been to find a Java plugin for the *#ifdef* directives preprocessing, in order to select the feature set and to instantiate the product line. The JavaPP preprocessor [7] has solved the lack of this plugin. On the other hand, the *SPL-TuPI* tool made possible to define the feature set and to instantiate the product line in an easy way, using the JavaPP preprocessor.

As future work, we intend to implement the validation of the constraints and the dependencies among features during the feature selection process, following also the XML standard from the FeatureIDE tool [5]. These restrictions will avoid the selection of invalid combinations.

References

1. Pohl, K., Böckle, G. and van der Linden, F.J. (2005) **Software Product Line Engineering:** Foundations, Principles and Techniques. Springer, NY.

2. Linden, F.v.d., Schmid, K. and Rommers, E. (2007) **Software Product Lines in Action:** The Best Industrial Practice in Product Line Engineering. Springer, NY.

3. Apel, S. et al. (2013) **Feature-Oriented Software Product Lines:** Concepts and Implementation. Springer, Berlin/Heidelberg.

4. Pereira, J., Constantino, K. and Figueiredo, E. (2014) A Systematic Literature Review of Software Product Line Management Tools. In Schaefer, I. and Stamelos, I.

(eds), **Software Reuse for Dynamic Systems in the Cloud and Beyond**. Lecture Notes in Computer Science, Springer International Publishing, pages 73-89.

5. Thüm, T. et al. (2014). FeatureIDE: An extensible framework for feature-oriented software development. **Science of Computer Programming**, 79, 70-85.

6. Batory, D. (2005) Feature models, grammars, and propositional formulas. In **Proc. of the 9th international conference on Software Product Lines**. pp. 7-20.

7. Kropf, J. (2010) **Java Preprocessor for Apache Ant**, http://git.slashdev.ca/javapp.

8. Spencer, H. and Collyer, G. (1992) **#ifdef Considered Harmful, or Portability Experience With C News**. Technical Report, University of Toronto, San Antonio, TX.

9. Machado, L. et al. (2014) SPLConfig: Product Configuration in Software Product Line. In **Brazilian Conference on Software: Theory and Practice - Tools Session**. pp. 85-92.

10. Cirilo, E., Kulesza, U. and Lucena, C.J.P. (2007) GenArch – A Model-Based Product Derivation Tool. In **SBCARS**. pp. 31-44.

11. Cirilo, E. et al. (2011) GenArch+: an extensible infrastructure for building framework-based software product lines. In: **Proceedings of the tenth international conference on Aspect-oriented software development companion**, pages 69-70. ACM.