Validating the existence of Design Patterns in UML models using First-Order Logic statements derived from GEBNF definitions

Richard Amphlett, Ian Bayley, Lijun Shan, Hong Zhu

In the Gang of Four's (GoF) book "Design Patterns: Elements of Reusable Object-Oriented Software"[1] the exact implementation of Design Patterns (DPs) is up to interpretation, because much of the subtlety of the DP structure is described in text. Given a stricter definition of a DP in GEBNF[2], can this definition be used to validate the correct use of DPs within UML models?

Introduction

If Software Engineering is to be regarded as an equal to the other engineering professions, then the quality of software produced by the Software Engineers needs to improve. Two of the solutions that have emerged to help achieve this goal are Unified Modelling Language (UML) and Design Patterns. UML provides a Model Driven Architecture approach to systems design. DPs provide general purpose solutions to commonly encountered design problems. Both are language and platform independent and, in the GoF's book, DPs are described in Object-Modelling Technique (OMT) a predecessor to UML.

The work for this URSS student scholarship uses as its basis the work completed by Bayley and Zhu in formalizing Design Patterns using GEBNF, a graphical extension to BNF, and the work of Shan in the formalization of UML semantics in first-order logic (FOL) [3]. Shan's work included the development of a UML model consistency checker LAMBDES; this tool uses the FOL theorem prover SPASS[4] as one of its main components

Approach

By extending LAMBDES to allow DP definitions (DPDs) to be included in the SPASS input files, UML models could be checked for the inclusion of DPs, *see LAMBDES-DP structure in Fig 1*.

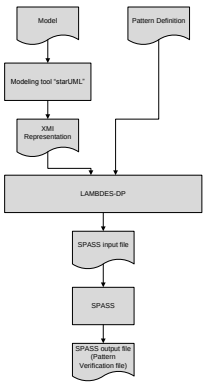

Figure 1. Structure of LAMBDES-DP

For this process to be successfully completed the GEBNF DPDs, *example in Fig 2*, need to be converted to the SPASS input format, *example in Fig 3*.

Components

- *AbstractClass* $\in$ *classes*
- *templateMethod* $\in$ *AbstractClass.opers*
- *others* $\subseteq$ *AbstractClass.opers*

Static Conditions

- *templateMethod.isLeaf*
- *templateMethod* $\notin$ *others*
- $\forall o \in$ *others.*$\neg$*isLeaf*

Dynamic Conditions

- $\forall o \in$ *others.callsHook(templateMethod,0)*

Figure 2. GEBNF Template Method DPD

```
%%% TEMPLATE METHOD FOL %%%
formula(exists([
  xAbstractClass,
  xTemplateMethod,
  xOthers],
and(

%%% COMPONENTS %%%
  Class(xAbstractClass),
  ownedOperation(xAbstractClass,xTemplateMethod),
  ownedOperation(xAbstractClass,xOthers),

%%% STATIC CONDITIONS %%%
  isLeaf(xTemplateMethod,bTrue),
  not(equal(xTemplateMethod,xOthers)),
  isLeaf(xOthers,bFalse)

%%% DYNAMIC CONDITIONS %%%
  callsHook(xTemplateMethod,xOthers)
)).
```

Figure 3. SPASS Template Method DPD

By converting all 23 of the GEBNF GoF DPDs in this way and by running LAMBDES-DP in batch mode it was possible to verify the existence of each GoF DP in a concrete UML model.

Experimental Results

To test this DP verification process, runs against a number of sets of UML models containing concrete instances of the 23 GoF DPs were completed. The test sets differed both in complexity and DP UML sources. Figs 4-6 give experimental output color coded, *Fig 7*, to highlight errors in the DP verification process. The time taken to verify each DP is marked in each cell (in seconds).

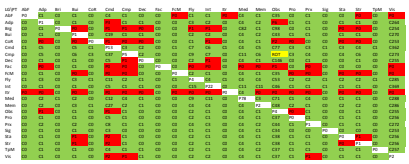

Figure 4. Minimal GEBNF

Figure 5. GoF OMT

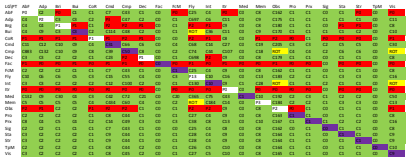

Figure 6. StarUML Wizard

Valid Positive	Valid Negative
False Positive	False Negative
Timeout after 16.5 minutes (SPASS configurable maximum)	

Figure 7. Color Key

Not all false positives imply errors. In the case of the minimal GEBNF test, *Fig. 4*, false positives indicate that the structure of one DP is a subset of the structure of another.

As the complexity of the UML models increases, as is the case in the GoF examples, the number of false positives increases due to accidental matches caused by additional none-DP classes included in the model e.g. client classes.

The false negatives in the StarUML examples (the UML design tool used to create all of the models) are caused by the StarUML design team's different interpretation of DP requirements. These interpretational inconsistencies arise because of the textual nature of some parts of the GoF DPDs.

Future Work

Due to the increasing number of timeouts that occurred as the complexity of the UML models increases, a more efficient DP verification process need to be designed if there is any hope of being able to detect DPs in real industrial ULM models. This may be achievable by more intelligent creation of the FOL input files, but may require the tool being implemented using a different logic and/or theorem prover. It is hoped that this work will be completed as part of a final year dissertation project.

References

1. Gamma, E., et al., *Design patterns: elements of reusable object-oriented software*. Addison-Wesley professional computing series. 1995, Reading, Mass. ; Wokingham: Addison-Wesley. xv, 395 p.
2. Zhu, H. and Bayley, I. *Specifying Behavioural Features of Design Patterns*, in *COMPSAC*. 2008, IEEE: Turku, Finland.
3. Shan, L. and Zhu, H. *A Formal Descriptive Semantics of UML*, to appear in Proc. of ICFEM 2008, Oct. 2008.
4. Geiß, V. *SPASS: An Automated Theorem Prover for First-Order Logic with Equality*. 2008 [cited 2008 09/09/2008]; Available from: http://www.spass-prover.org/.