Software Transformations: A formalism to trace program modifications

Mathieu Lemoine
Université de Montréal, DIRO,
lemoinem@iro.umontreal.ca

Yann-Gaël Guéhéneuc
École Polytechnique de Montréal, Software Engineering,
yann-gael.gueheneuc@polymtl.ca

2009/01/16
(4th MoSART Meeting)

1. Introduction
   - Context
   - Running Example
2. Background
3. Transformations: A Mathematical Formalism
4. Implementation
5. Future Work and Conclusion
   - Future Work
   - Conclusion
6. Bibliography

1 Introduction
- Context
- Running Example

2 Background

3 Transformations: A Mathematical Formalism

4 Implementation

5 Future Work and Conclusion
- Future Work
- Conclusion

6 Bibliography

Typical Problem: Documenting Software Evolution

Many people working at many abstraction levels on the same project at the same time.

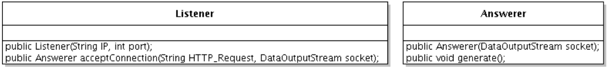
Problem: How to keep track of each modification, in a way readable by every one (Developer, Analyst, Manager)?

Goal: reduce cost, ease communication, trace software evolution.

Current Solutions

- Versionning Control Repositories (CVS, SVN, DARCS, ...)
  $\Longrightarrow$ textual documents, only for developer
- Model Driven/Reverse Engineering Architecture Tools (OMONDO, Ptidej [Gué05], ...)
  $\Longrightarrow$ no simultaneous modifications of code and model.
- post-mortem analysis (detection of refactorings, entity matching [ACPT01])
  $\Longrightarrow$ no live feedback, no rollback capabilities.

Development of a web server - Model



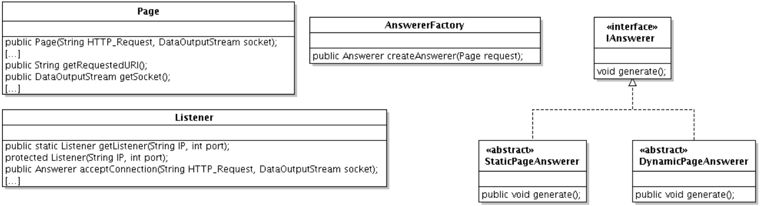
Listener

public Listener(String IP, int port);
public Answerer acceptConnection(String HTTP_Request, DataOutputStream socket);

Answerer

public Answerer(DataOutputStream socket);
public void generate();

First-draft, orignal specification



Page

public Page(String HTTP_Request, DataOutputStream socket);
[...]
public String getRequestedURI();
public DataOutputStream getSocket();
[...]

AnswererFactory

public Answerer createAnswerer(Page request);

«interface»
IAnswerer

void generate();

Listener

public static Listener getListener(String IP, int port);
protected Listener(String IP, int port);
public Answerer acceptConnection(String HTTP_Request, DataOutputStream socket);
[...]

«abstract»
StaticPageAnswerer

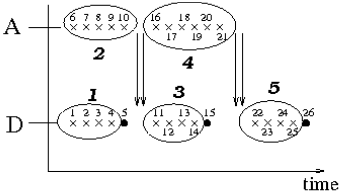
public void generate();

«abstract»
DynamicPageAnswerer

public void generate();

Actually created code modelization

Development of a web server - Time line



A is Analyst, D is Developper, Each little number is a single modification.

1. Modification of Listener constructor (code modifications)
2. Creation of Item class (model modifications)

   Modifications sent to D
3. Renaming of Item to Page and implementation of Page (code modifications)
4. Creation of IAnswerer interface and its Factory (model modifications)

   Modifications sent to D
5. Implementation of the IAnswerer hierarchy (code modifications)

1 Introduction
- Context
- Running Example

2 Background

3 Transformations: A Mathematical Formalism

4 Implementation

5 Future Work and Conclusion
- Future Work
- Conclusion

6 Bibliography

Hoare Triples [Hoa69]

Representation of action having

- guards (pre-condition)
- effects (post-condition)

Notation:
“{pre-condition}action{post-condition}”

Group and Homomorphism [DF04]

- Group:
  Set of mathematical objects with an intern operation $\circ$.
  Properties:
  - $\circ$ is associative,
  - There is a unique neutral element for $\circ$,
  - Each element has an unique inverse.
- Group Homomorphism:
  Mathematical function from one group to another preserving the group structure: $F(r_1 \circ r_2) = F(r_1) \circ F(r_2)$.

1 Introduction
- Context
- Running Example

2 Background

3 Transformations: A Mathematical Formalism

4 Implementation

5 Future Work and Conclusion
- Future Work
- Conclusion

6 Bibliography

Vocabulary: What is a transformation?

Transformation :

- Mathematical and reifiable object.
- Modification of a model (software representation).
- Defined under a meta-model (model specification [BP01]).

Notation $\mathbb{T}_{MM}$: set of all transformations under the meta-model MM.

Our goal: Transpose them between meta-models.

Example: Renaming a class could be a transformation.

Transformations as Hoare Triple

Transformation = Modification:

- Is an action,
- Depends on a previous model state,
- Creates a new model state

Example:

$$\{\exists C_0 \wedge \nexists C_1\}\text{Rename } C_0 \text{ in } C_1\{\nexists C_0 \wedge \exists C_1\}$$

Set of Transformations as Groups

$\circ$ is sequencement ("followed by", "then").

- Internal Operation:
  $\forall a, b \in \mathbb{T}_{MM}, a \circ b \in \mathbb{T}_{MM}$
- Associativity:
  $\forall a, b, c \in \mathbb{T}_{MM}, (a \circ b) \circ c = a \circ (b \circ c) \overset{\text{def}}{\equiv} a \circ b \circ c$
- Unique Neutral Element (Identity):
  $\exists! \mathbb{I} \in \mathbb{T}_{MM}$ st $\forall a \in \mathbb{T}_{MM}, a \circ \mathbb{I} = \mathbb{I} \circ a = a$
- Unique Inverse:
  $\forall a \in \mathbb{T}_{MM}, \exists!\ a^{-1} \in \mathbb{T}_{MM}$ st $a \circ a^{-1} = a^{-1} \circ a = \mathbb{I}$
- Inversion of sequence:
  $\forall a, b \in \mathbb{T}_{MM}, (a \circ b)^{-1} = b^{-1} \circ a^{-1}$

Example: The reverse of Renaming a class *FOO* as *BAR*, is to rename the class *BAR* as *FOO*.

Transpositions as Group Homomorphism

Transformations are elements of Groups,
therefore
Transpositions are Group Homomorphisms.

$$F(a \circ b) = F(a) \circ F(b)$$

$$F(\mathbb{I}_{MM1}) = \mathbb{I}_{MM2}$$

$$F(a^{-1}) = F(a)^{-1}$$

Example: There would be a transposition between the code, the developper work on, and the model, the analyst work on.

Commutativity

commuting transformations = change modifications order.

Example: “Renaming *FOO* in *BAR*, then Adding a method *baz* in *BAR*” becomes “Adding a method *baz* in *FOO*, then Renaming *FOO* in *BAR*”.

1 Introduction
- Context
- Running Example

2 Background

3 Transformations: A Mathematical Formalism

4 Implementation

5 Future Work and Conclusion
- Future Work
- Conclusion

6 Bibliography

Implementation

- PADL [Gué03, AA03, GA08]
  - Meta-model used to represent specification of programs,
  - High-level models.
  - Developed to represent patterns and abstract designs,
- JCT
  - Meta-model used to represent program code source,
  - Low-level models (Bound Abstract Syntax Tree),
  - Developped to represent Java Program, similarly to javac.

1. Introduction
   - Context
   - Running Example
2. Background
3. Transformations: A Mathematical Formalism
4. Implementation
5. Future Work and Conclusion
   - Future Work
   - Conclusion
6. Bibliography

Future Work

Implementation in progress:

- JCT implementation almost finalized,
- PADL and JCT transformations implementation in progress,
- Transposition between JCT and PADL to specify and implement,
- PADL and JCT transformations commutativity implementation in progress.

Conclusion

Our approach provides:

- Mathematical theory, verifiable, formal.
- Live feedback and concurent modifications of the program, at many levels of abstraction.
- Reversibility (Rollback facilities).
- Traceability of each transformations.

But is purely theoritical now. Implementation in progress.

Bibliography I

[AA03] Hervé Albin-Amiot, *Idiomes et patterns java : Application à la synthèse de code et à la détection*, Ph.D. thesis, université de Nantes, février 2003.

[ACPT01] G. Antoniol, B. Caprile, A. Potrich, and P. Tonella, *Design-code traceability recovery: selecting the basic linkage properties*, Sci. Comput. Program. **40** (2001), no. 2-3, 213–234.

[BP01] Jean Bézivin and Nicolas Ploquin, *Tooling the MDA framework: A new software maintenance and evolution scheme proposal*, Journal of Object-Oriented Programming **14** (2001), no. 12, .

[DF04] David S. Dummit and Richard M. Foote, *Abstract algebra*, third ed., Wiley, 2004.

Bibliography II

[GA08] Yann-Gaël Guéhéneuc and Giuliano Antoniol, *Demima: A multi-layered framework for design pattern identification*, Transactions on Software Engineering (2008), (english), *Accepted for publication.*

[Gué03] Yann-Gaël Guéhéneuc, *Un cadre pour la traçabilité des motifs de conception*, Ph.D. thesis, École des Mines de Nantes et Université de Nantes, juin 2003.

[Gué05] Yann-Gaël Guéhéneuc, *Ptidej: Promoting patterns with patterns*, Proceedings of the 1st ECOOP workshop on Building a System using Patterns (Mohamed E. Fayad, ed.), Springer-Verlag, July 2005 (english).

[Hoa69] C. A. R. Hoare, *An axiomatic basis for computer programming*, Commun. ACM **12** (1969), no. 10, 576–580.

The End!

Thank You!

Question?