NOREX: A Distributed Reengineering Environment

Mihai Balint[1], Petru Florin Mihancea[1], Tudor Gîrba[2] and Radu Marinescu[1]

[1]LOOSE Research Group, Politehnica University of Timişoara, Romania
[2]Software Composition Group, University of Bern, Switzerland

Abstract

Several reengineering environments have been created to provide for a unified infrastructure in which various approaches can be employed together. While the collaboration between tools is very strong within such environments, currently the inter-environmental collaboration is very weak and happens mainly at the level of data-files exchange. Consequently, the different groups of researchers are only collaborating shallowly via data, rather than at the level of analysis. In this demo, we present NOREX*, a distributed reengineering environment that allows different groups of researchers to transparently use and combine existing techniques, and share their own, transcending any parochial barriers (e.g., implementation language or environment).*

1 Introduction

Reengineering is a complex task that requires several techniques to be employed together. Over the years, many reengineering analyses have been proposed and implemented. Several environments have been created in the past to provide a unified infrastructure in which various approaches can be employed together [5]. Tthe authors are involved in the development of two such environments: iPlasma [4] and Moose [5].

Nevertheless, these environments are rather isolated from one another, as the communication between environments is confined to data transfer via exchange formats [2]. The main cause of this isolation resides in the fact that environments are oftentimes implemented in different programming languages, and even when they are implemented in the same language they have a dedicated infrastructure (*e.g.,* different meta-models, different front-ends).

There have been some attempts at making analyses defined on different metamodels work together. For example, Dean *et al.* present an attempt at running an analysis on an instance of another conceptually equivalent metamodel using ontologies [3]. While their attempt is more about connecting tools by matching the underlying metamodels, in this demo we present *NOREX*, a distributed environment that extends the idea of an integrated environment adding the benefits of distributed services and as such enabling large-scale community-based reengineering and reengineering research.

```
public Integer NoPM(
        Object class,
        CollectionCommand methods,
        BooleanCommand isPublic) {
    int publicCount = 0;
    for (Object method: methods.invoke(class))
        publicCount ++;
    return publicCount;
}
```

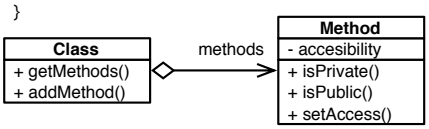

Figure 1. Number of Public Methods

2 Reengineering with *NOREX*

NOREX provides several distinct services, ranging from data transfer and remote execution in a programming language independent manner to metamodel support and analysis registration services. It achieves this using a client-server architecture and *Web Services* as a means of communication.

***NOREX* Services and Servers.** The central element of the environment are *NOREX services*. These reside on *NOREX servers* and contain the implementation of reengineering techniques designed for a certain type of model entity (*e.g.,* classes, methods). A key element of the approach is the low coupling with the metamodel, which is achieved by insuring that all data accesses are wrapped within *command objects*. These command objects are passed as parameters to the service. The *NOREX* server provides the metamodel description in form of a Meta Object facility (MOF) [1] instance and facilities for executing of a set of reengineering techniques defined in terms of that metamodel. As an example we present the implementation of a service that

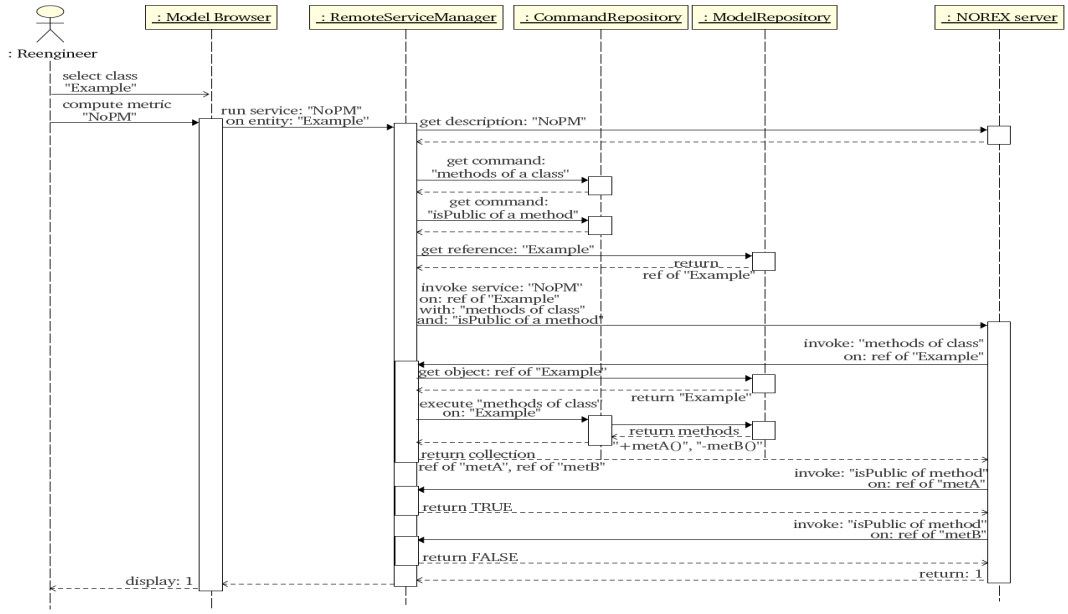

Figure 2. Invoking Number of Public Methods

implements the metric entitled *Number of Public Methods* (NoPM) applied to class entities (Figure 1). The service is implemented as any regular method. It receives three parameters: (1) the class to be measured, (2) a command that returns a collection containing the methods of the class, and (3) a command that returns the public status of a method.

***NOREX* Clients.** Clients use the service provided information to build, navigate and execute services on models of the target software system. To use *NOREX* services, the client side needs to fulfill a number of requirements, which are *de facto* the implementation of the following tools: (1) a metamodel generator, (2) a model builder, (3) a model browser, and (4) a service runner.

Step 1: Generate the Metamodel. Let's consider again the metamodel represented in the diagram of Figure 1. MOF provides the necessary facilities for specifying metamodels and it can be used to generate metamodel implementations, in this case Java or Smalltalk classes representing Class and Method entities.

Step2: Build the Model. Once generated, the metamodel and the *NOREX builder services* enable the creation of models of software systems on which reengineering activities are to be performed. Currently the *NOREX* environment provides builder services working for C++, Java, Smalltalk, or C#. These services are used to parse a set of source code files and build at the clients side an objectual model of the code, in conformance with the metamodel with which the builder service is associated.

Step 3: Browse the Model. Model browsing or navigation in a *NOREX* client is important from both the control and comprehension point of view. A model browser provides the means to inspect and navigate a system model, given that the description of the metamodel for that model is provided as a MOF instance. For example, given the metamodel described in Figure 1, from Class we can navigate to methods.

Step 4: Execute a Service. The actual execution of a *NOREX* service is a matter of providing the entities and the required command objects. To avoid large transfers we use a referencing mechanism that ensures that no model entity will leave the client. After service execution finishes the result is returned to the client and it is passed to the client's tools for further processing. To illustrate the execution of a service we depicted in Figure 2 how the computation of the Number of Public Methods (NoPM) metric. Other services have been defined for a range of software metrics, visualizations and design flaw detection strategies.

References

[1] O. M. Group. Meta object facility (MOF) 2.0 core final adopted specification. Technical report, Object Management Group, 2004.

[2] R. C. Holt, A. Winter, and A. Schürr. GXL: Towards a standard exchange format. In *Proceedings WCRE '00*, Nov. 2000.

[3] D. Jin and J. R. Cordy. Integrating reverse engineering tools using a service-sharing methodology. In *Proceedings of ICPC'06*. IEEE Computer Society, 2006.

[4] C. Marinescu, R. Marinescu, P. F. Mihancea, D. Ratiu, and R. Wettel. iplasma: An integrated platform for quality assessment of object-oriented design. In *ICSM (Industrial and Tool Volume)*, pages 77–80, 2005.

[5] O. Nierstrasz, S. Ducasse, and T. Gîrba. The story of Moose: an agile reengineering environment. In *Proceedings of the European Software Engineering Conference (ESEC/FSE 2005)*, pages 1–10, New York NY, 2005. ACM Press. Invited paper.