Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Incremental dependency analysis over large software infrastructure

Author
Donat Csikos

Supervisors
dr.Istvan Rath, Akos Horvath

October 11, 2012

Contents

Chapter 1

Introduction

1.1 Control systems at CERN

The European Organization for Nuclear Research known as CERN is an international organization which runs the the world's biggest particle physics laboratory. Established at 1954 and situated on the Swiss-French border next to Geneva. The organization's main goal to operate particle accelerators (such as the Large Hadron Collider) and all the necessary infrastructure.

Obviously not only physicists work az CERN: numerous scientist and engineers are working on the design and the maintenance of the software and hardware equipments in a well-defined structure. The Controls Group is responsible for – among the others – the design and implementation of both software and hardware used in the controls systems. This involves providing software frameworks, providing alarm systems and communication middleware and numerous related tasks.

After the LHC became operational the number one objective for the Controls Group is to maintain the operation uninterrupted without any downtime. This implies that a upgrading certain parts of the system should not interfere with other component's.

The controls systems is a complex, distributed and highly modular system which has three tiers. On the top there are the GUI applications which are written in Java. The middle or business layer is also consists of Java applications. On the lowest or hardware level there are C/C++ applications running on real-time systems. Maintaining a complex system like this requires a set of special approaches provide desired quality and maximize the availability of the systems in one.

1.2 Main goal: Smooth upgrades

Smooth upgrade means when a software is being updated than all the other softwares which depend in the upgraded one will remain operational. A basic example of braking this principle, when the newer version is binary incompatible with dependant softwares and cause load-time error.

There are a variety of tools and methods how to achieve smooth upgrades. For instance one can introduce manual procedures and protocols to the development lifecycle.

Another possibility to enforce some code metrics to achieve this goal.

The last approach – which my solution tries to implement – is the concept of *incoming dependencies*. As an example let's see a developer who is working a Java library which happens to be used by some clients. Now let's say there is an error in the bug which requires a change in the library's API for the resolution. Our developer obviously wants to know who is using that part of the API. The users (as in code parts) are considered as incoming dependencies.

My solution which I present in this paper is a possible implementation for querying incoming dependencies. The first valid question is: why didn't we just reuse an existing implementation? The answer is: because there is no one. There are specialities about the situation my tool has to solve.

Size limitation If we are able to load every single jars into an IDE the problem would be solved: we could use the internal navigation features to navigate through the dependency graph. But because we have thousands of jars (and tens of thousands if we count the possible versions) it is evidently impossible.

Fine-graininess of dependencies The existing tools usually cannot determine the desired dependency resolution level.

No source code analysis We don't want to do a source code analysis because it would imply a dependency resolution for concerned projects every time a new version of a product comes out. This requirement eliminates a number of possible tools.

Because of these reasons we decided to implement our own solution called Dependency Analysis Tool. In the following chapters I will present the design and the implementation for this tool and how is it currently applied az CERN Controls Group.

Chapter 2

Background

2.1 Related technologies

Before proceeding to the main part of this paper, I am going to give an overview about the related technologies used by my solution. I Why do we have to present the related technologies.

2.1.1 Java byte code specification

How the java vm works. What is the input. What kind of information can be extracted from here.

2.1.2 Apache Commons Byte Code Engineering Library

[1] Abstraction over Java byte code specification. How it works. Create diagram! Possible use-cases.

2.1.3 Eclipse

2.1.4 Eclipse Integrated Development Environment

The Eclipse Project [2] is an open source software development project dedicated to providing a robust, full-featured, commercial-quality, industry platform for the development of highly integrated tools. It was developed by IBM from 1999, and a few months after the first version was shipped, IBM donated the source code to the Eclipse Foundation.

The Eclipse project consists of many subprojects, the most important being the Eclipse Platform, that defines the set of frameworks and common services that collectively make up „integrationware" required to support a comprehensive tool integration platform. These services and frameworks represent the common facilities required by most tool builders, including a standard workbench user interface and project model for managing resources, portable native widget and user interface libraries, automatic resource delta management for incremental compilers and builders, language-independent debug infrastructure and infrastructure for distributed multi-user versioned resource management.

The Eclipse Platform has an easily-extendable modular architecture, where all functionality is achieved by plugins, running over a low-level core called Platform Runtime. This runtime core is only responsible for loading and connecting the available plugins, every other functionality, such as the editors, views, project management, is handled by plugins. The plugins bundled with Eclipse Platform include general user interface components, a common help system for all Eclipse components, project management and team work support.

2.1.5 Eclipse Modeling Framework

Copy it from the stock text.

2.1.6 EMF-IncQuery

Incremental model queries over emf models. Documentation site.

2.1.7 Other related technologies

Spring Framework

To develop modular, testable and configurable applications.

Maven

Build system

Tycho

Maven extension to build eclipse plugins.

Oracle database

Commercial relational database management system. Full sql support. Extensive feature set, one of the market-leaders.

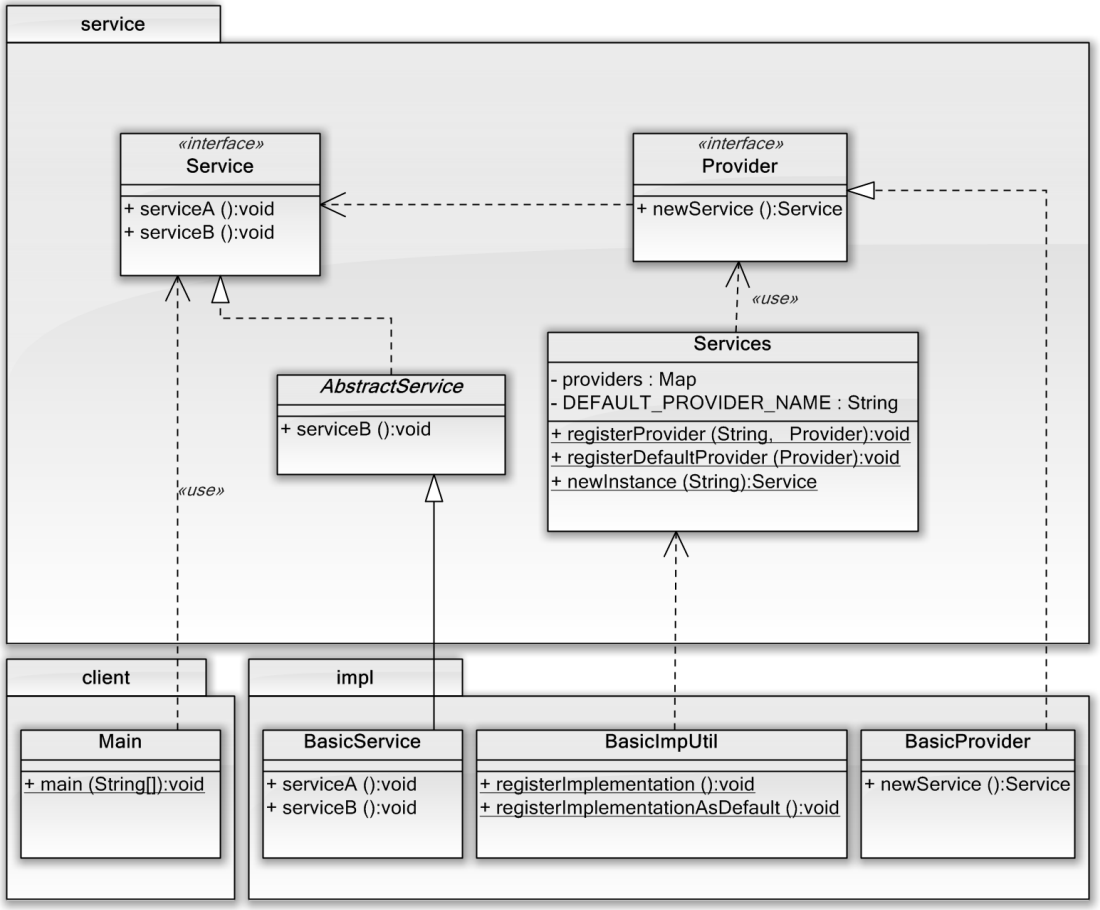
2.2 Example: Service Provider Framework

The service provider framework is a possible use-case for the Adapter design pattern. The implementation I am going to present derived from the book [3] Effective Java.

Overview of the pattern. Three different people

Description of the classes one-by one. Possible source code can be inserted here.

Figure 2.1: *Service Provider framework example*

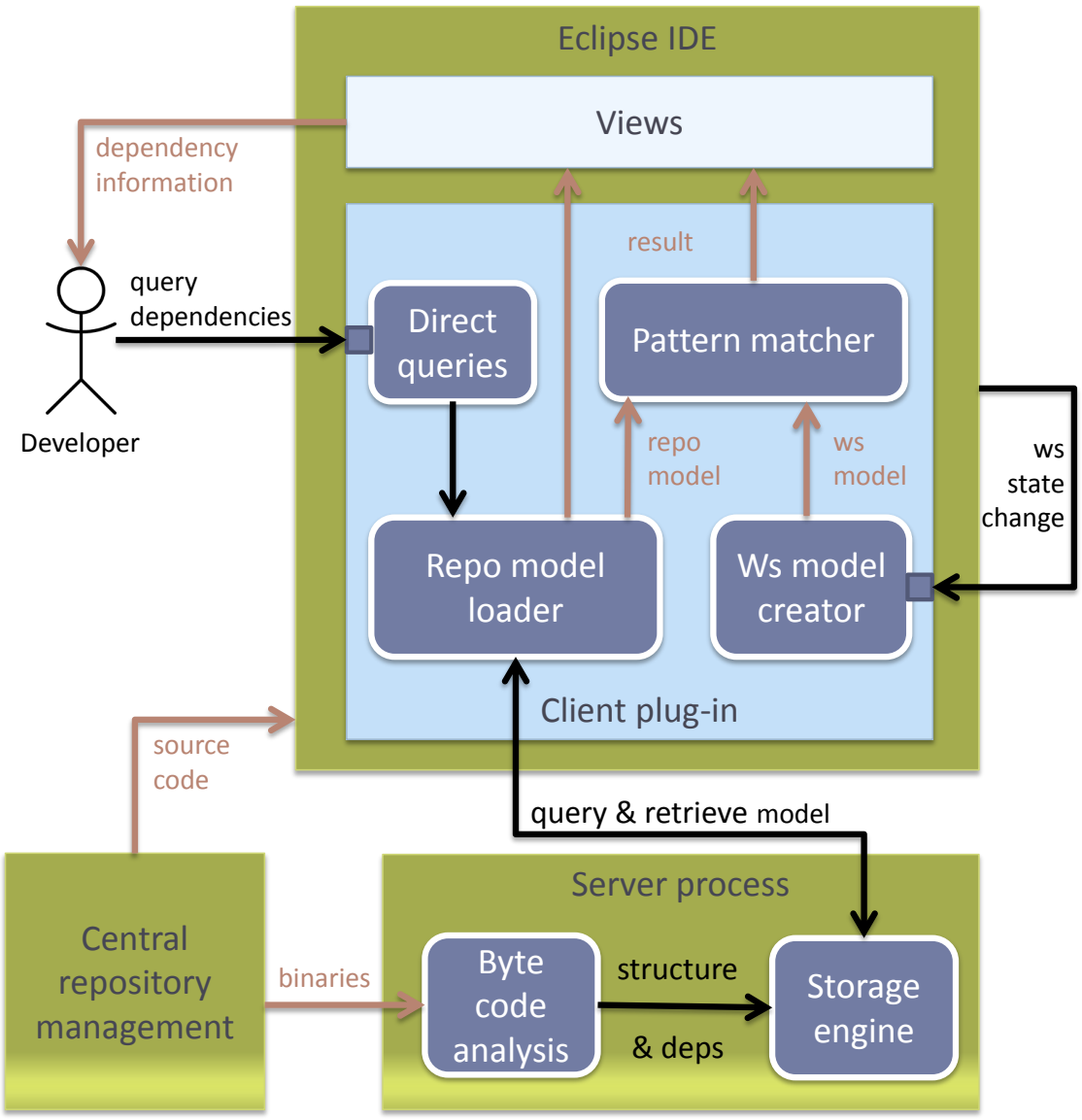


Chapter 3

Overview

The tool aims to find incoming dependencies as it was described in the previous chapter. Two sentenced about the main goal.

Figure 3.1: *Overview of the implemented system*



The overview of the implemented software is on figure. 3.1 The system is implemented in two steps. First the CERN part. Direct queries. The second was done in the university. Model loading and creation and pattern matching with EMF-IncQuery. I will detail it later.

The first element on the figure to discuss is the „Central repository management". CERN infrastructure contains an Ant-based software called Commonbuild. Commonbuild's covers the common tasks with softwares such as building, resolving dependencies, releasing and deployment. As a result of using Commonbuild we have a well-maintained source code repository (released versions as svn tags) and directory of the released binary files with a good description.

A typical workflow. Developer checks out a source code from the SVN and starts to work on it. For querying the dependencies he has 2 options. Direct queries or using the pattern matching. A direct query means.. A pattern means. At every save. Fast but requires more memory. Trade-off.

The direct queries was developed at CERN. The pattern-based is an extension for this work

The server side Upon somebody does a release in the repositories, the server process finds the correspondent released binaries (jar format) and tries to map the structure and dependencies using bytecode analysis. In order not to fiddle parsing binary files the server utilizes the Apache BCEL. The structure and the dependencies are passed to the storage engine which defines a set of operations to find, store and retrieve certain subset of the model.

The client side The „Direct queries"' module enables the developer to execute queries which return the. As soon the result comes back from the server it is displayed in a view for further analization.

The extension is a pattern matching solution. The "Ws model creator" generates an incrementally updates an emf instance model (driven by the workspace changes). Also an emf instance model describing the structure and the dependencies in the repository is loaded from the server process.

The two emf model serves as an input for the EMF-IncQuery engine. The IncQuery loads the models and tries to find more refined dependency information by applying a set of patterns on the models.

Chapter 4

Details

4.1 Infrastructure at CERN controls systems

4.1.1 Used tools

SVN, commonbuild, jira; check this at some papers of vito.

4.1.2 Development workflow

Original idea comes from the typical development workflow lifecycle at CERN control systems.

4.2 Bytecode analysis

we don't want to work with the source code, because the size of the problem.
what kind of information can be extracted from the bytecode
IMportant: this section heavliy relies what details are covered in the introduction part.

4.3 Persisting dependencies

4.4 Direct queries

4.5 Repository EMF model

4.6 Creation of workspace EMF model

4.7 Pattern matching

Chapter 5

Evaluation

5.1 Functional evaluation

5.2 Performance evaluation

Chapter 6

Conclusion and Future work

I created a tool. It can be useful for a library developer wqorking at large, software-oriented organization where lots of inter-depending software are developed and maintained. Becomes extremely useful when the concept of smooth upgrades is essential.

Implemented a precise, source-based query solution along with a not entirely precise but fast and reactive display of dependency data.

As for the future there are plenty of directions to go: Slicing the repository model to load just the required part of it. Notification about the repository change. Define and extract more detailed dependencies from the binaries. Extend the dependency resolution to c++ software.

Bibliography

[1] Apache bcel documentation. `http://commons.apache.org/bcel/manual.html`. Accessed 08.10.2012.

[2] Eclipse project website. Date retrieved: October 10, 2012.

[3] Joshua Bloch. *Effective Java (2nd Edition) (The Java Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2 edition, 2008.