Communication Test for Object-Oriented Systems using Gossiped Data (Fast Abstract)

Robert Kalcklösch and Peter Liggesmeyer

Software Engineering: Dependability Group
TU Kaiserslautern
[kalckloesch|liggesmeyer]@informatik.uni-kl.de

1 Introduction

Today, computer systems rarely consist of only few components or classes in object-oriented software, respectively. For complex systems like enterprise solutions the number of classes often exceeds ten thousands with a lot more instantiated objects during the lifecycle of the system. In order to fulfill the desired purpose, the objects have to communicate with each other. Apparently, the communication is not an end in itself, but rather an integrated part in the business processes of the system. Hence, communication is a crucial part for the proper operation of every complex software system and should be carefully tested.

In this paper we propose a solution for the integration of complex object-oriented software written in Java placing emphasis on the test of communications between classes.

2 Communication testing

2.1 Gossip Protocol

In the real world, gossip is used to spread information about specific group members throughout a group of several connected entities. Within the test setting the information spread consists of all interactions and their outcomes involving all classes known to the two entities currently interacting with each other. Thus, each class maintains a database with information about all other classes she knows of and the outcomes of all interactions she hears about or is directly involved in.

The actual gossiping follows a specific protocol: Whenever two classes interact with each other (e.g. one object belonging to one class calls a method or service of an object belonging to another class), the caller rates[1] the outcome of the interaction and stores the result in her database. Afterwards, both components exchange their databases. In fact, they gossip about all other components they know about. After the exchange, both components integrate the new data in their own databases.

2.2 Communication testing using the Basic Protocol

Starting from the basic gossip protocol, a given system can be easily tested. The basic protocol is applied to each and every interaction that takes place within the system to be tested. With progressing test duration classes learn about other classes in the system they have not yet interacted with. Thus,

[1] This rating is currently based on the presence of an exception. But as the violation of assertions also results in exceptions, not only run-time errors could be identified.

the local database contains the knowledge a class has over the system. It represents the subjective view for a given time t and is a subset of the objective system state containing all interactions from the beginning till t.

3 Code Integration

In order to allow software systems to gather information about the communication links between their classes, the functionality of each class has to be enhanced. Therefore, we have developed a specific code injector based on AspectJ which extends each class with the following parts: An internal memory for the observed interactions (`ErrorMemory`), a method which stores the information in the internal memory (`rateInteraction(...)`), a method for the exchange of all stored information (`exchangeMemory(...)`), a method which checks the exchanged information for new data and incorporates it in the local memory (`incorporateData(...)`), and a method which stores the actual information (i.e., a snapshot of its local memory) periodically in a persistent manner (i.e., *.log* files) (`safeData()`).

The original functionality is not modified, even if the system contains aspects itself. Through the use of AspectJ the code injector is able to inject code directly into java-byte-code (i.e., *.jar* archives). Therefore, it is possible to separate the actual test from the development environment by simply handing the compiled system to the testers.

The only impact the injected code has, is an increase of processing time. The additional code takes some time to process and, hence, the execution time is much longer. Therefore, it is not suitable for real-time applications. The exact impact on the execution time needs to be quantified in further studies.

During the test, each class stores its local memory periodically in a persistent manner. There is one log-File for each class in the system, consisting of serialized Hashmaps. This huge amount of data has to be evaluated, so that classes containing the most 'problems' could be easily identified. Therefore, we developed a tool, which transfers this data into a representation of a three dimensional graph. This graph could then be processed by a 3D-Graph Viewer, in our case Wilmascope.

4 Conclusion and Outlook

We have shown that our approach of gathering information about the communications within an object-oriented software system with a gossip protocol is possible. We provided a method to inject the necessary code in an existing software system without changing the functionality of the original system. Additionally, we provide a tool, which transforms the gathered data in a three dimensional graph representation, accessible by Wilmascope, which is used for displaying the results.

As this is currently a prove of concept, there are many issues that have to be addressed. First of all, the approach has to be tested in a real industrial project with at least several hundred classes. Also, we have to measure the exact impact of the injected code, on the execution time. Another point we have to further investigate is the visualizing of the gathered information. Currently, we display all edges and nodes in the graph, were it might be suitable to only display faulty ones. Also, the coloring is not settled yet. Last but not least, we have ongoing work to migrate the system to support other programming languages like C++, C#, and others.