RoboCup - UTTP Player Re-factoring Case Study

Bc. Peter Kajsa, Bc. Michal Kasan, Bc. Branislav Kontúr, Bc. Dávid Kováč, Bc. Martin Nepšinský, Bc. Tibor Sekereš*

Slovak University of Technology
Faculty of Informatics and Information Technologies
Ilkovičova 3, 842 16 Bratislava, Slovakia
TPteam06@googlegroups.com

Extended abstract

This extended abstract provides a case study focused on the re-factoring and modularization of a software complex agent consisting of many behaviors. We focused on the architecture of the ***RoboCup*** soccer player addressing non-functional requirements on the system. We describe our experience with re-factoring of a large unstructured code into modular behavior-based architecture.

The target of re-factoring was the last year's team Loptoši [2]. The source code of the player was platform independent. We decided to improve the structure of this player without changing functionality. From the object-oriented design viewpoint, player's architecture is not very suitable. Major part of the player consists of only three large classes (*PlayerTactics/GoalieTactics extends PlayerSkills extends PlayerKernel*). For example the class PlayerTactics has about 3500 lines of code and contains many comments like: "What is this doing?" The functionality is then very hardly understandable and changing the source code often leads to unpredictable behavior. The inheritance was not used properly in this case. The use of composition would be better in this case and the player should be split into more clearly understandable modules, not just three classes with unclearly roles. We also made some quality analysis of the source code. According to these analyses, the source code has several serious problems, e.g. cyclomatic complexity more than 10, different count of operators new and delete in one file, more than one return point per one function. Another serious problem is low understandability and therefore high effort needed for adding of new functionality or improving the existing one, because high effort is necessary for analyzing and studying this complex and ill structured code.

* Supervisor: Ing. Marián Lekavý, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava.

IIT.SRC 2008, Bratislava, March 21, 2008, pp. 1-2.

We decided to re-structure the player based on behaviors. We started our work by analysis and identification of behaviors in the source codes and in the documentation of previous players. The analysis was the most important part of our work, because there were several undocumented behaviors in the code and many of the documented behaviors were implemented inconsistent with the documentation or not implemented at all. As a result, we created documentation for all identified behaviors [1]. The documentation was created prior to the new code, thus serving as specification.

The re-factoring started from 3 classes with thousands lines of codes. The whole high behavior is placed in only two large methods. Both methods consist of many nested *if-then-else* blocks which define the behavior in all game situations. Every block consists of tens to hundreds code lines and there are a lot of state variables, containing the state of the player and the world. There is no visible structure of the code and the code is very complex. This makes the code un-modifiable and un-maintainable.

We separated individual behaviors and encapsulated them into individual modules. At the same time, the usage of other modules has to be declared explicitly in the module definition. This way, we achieve high level of modularization, loose coupling between the modules and thus better maintainability.

We chose bottom-up approach to the re-factoring. This way, we start with simple behaviors and build more complex behaviors by combining simpler ones. This approach showed to be better than top-down, because the behaviors at the top level were too complex and coupled with most of the lower parts of the code. Using top-down approach would require analyzing the whole code to separate a single top-level behavior. Separating a low-level module is simpler, because there are just a few interactions with the remaining code. When we reach the top level behaviors (with the bottom-up approach) , we can already build on the well-structured lower behaviors.

The new behavior-based structure allows easier and faster orientation in the code than the previous unstructured code. Further modifications should be easier with the new structure, because most changes are localized in one or a few behaviors. Therefore, this kind of changes can be made without affecting of other behaviors. There could be a problem if we change the behavior's interface, but with the loose coupling and explicit couplings between modules, this can be done with low effort.

The new structure of the player will allow other teams to focus on new strategies and improvement of behaviors rather than spending most time for analyzing and studying of complex and chaotic code.

Acknowledgement: This work was partially supported by the Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava.

References

[1] Kontúr, B., et al.: RoboCup – nové stratégie, Bratislava, FIIT STU, 2007, Team project - UTTP.

[2] Kútny, M., et al.: RoboCup – nové stratégie, Bratislava, FIIT STU, 2006, Team project - Loptoši.