# COORDINSPECTOR: a tool for extracting coordination data from legacy code

Nuno F. Rodrigues
DI-CCTC, Universidade do Minho
4710-057 Braga, Portugal
Email: nfr@di.uminho.pt

Luis S. Barbosa
DI-CCTC, Universidade do Minho
4710-057 Braga, Portugal
Email: lsb@di.uminho.pt

***Abstract*—Current software systems rely on, more and more, non trivial coordination logic for combining autonomous services typically running on different platforms and owned by different organizations. Often, however, coordination data is deeply entangled in the code and, therefore, difficult to isolate and analyse separately.**

**COORDINSPECTOR is a software tool which combines slicing[1] and program analysis techniques to isolate all coordination elements from the source code of an existing application. Such a reverse engineering process provides a clear view of the actually invoked services as well as the orchestration patterns which bind them together.**

**The tool analyses Common Intermediate Language (CIL) code. CIL being the native language of the Microsoft .Net Framework, COORDINSPECTOR is therefore suitable for processing code developed in any programming language compiling to the .Net Framework. The tool generates graphical representations of the coordination layer together with business process orchestrations specified in Orc.**

## I. INTRODUCTION

Intro

## II. IMPLEMENTING COORDINSPECTOR

COORDINSPECTOR[1] is a software analysis tool developed as a proof-of-concept of the ideas presented in this paper.

The tool, a snapshot of which is presented in Fig. targets CIL code, the native language of the Microsoft .Net Framework, to which every .Net compilable language ultimately gets translated to before being executed by the framework. This decision to target CIL code was not an arbitrary one. Indeed we intended the tool to be able to cope with as many programming languages as possible, because most real world software systems are developed in more than one language. Moreover, given the potential of the tool to assist legacy systems evolution, the "language agnostic" feature became an important invariant. Thus, by choosing CIL, the tool is presently able to analyse more than 40 programming languages[2], and this number has only but potential to increase.

In order to take advantage of existing CIL analysis tools, COORDINSPECTOR is developed as a plug-in for the CIL decompiler .Net Reflector[3]. The only, and important component

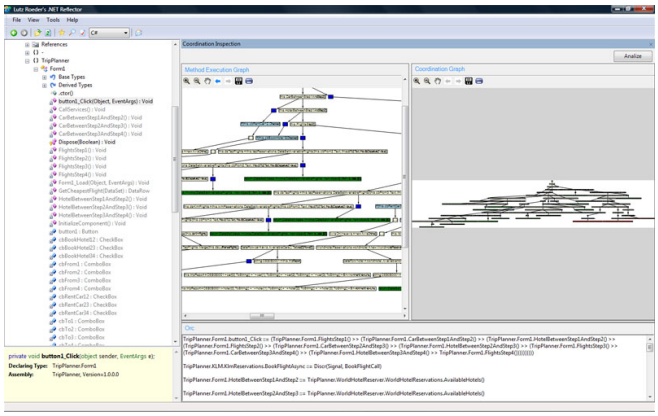

Fig. 1. COORDINSPECTOR

COORDINSPECTOR takes from .Net Reflector is the parse for CIL code which delivers an object tree representation of the CIL concrete syntax tree.

Such tree is then processed to build the corresponding MSDG instance. Given the intrinsic modularity of this process, it is executed by different components that are responsible for the calculation of each of the MSDG sub-graphs i.e., the MDG, ClDG, IDG and the NDG, as detailed in section **??**. Each component traverses the concrete syntax tree, using the object oriented proxy pattern, and collects the relevant information for the construction of a particular graph.

When applied to real world systems, and if executed sequentially, the MSDG calculation process can be a time consuming task because of the size and computational complexity involved. In order to cope with this situation one has improved the MSDG calculation performance by multithreading the tasks which build each MSDG sub-graph. This improvement reduced the MSDG calculation time to roughly on third of the original time.

The CDG calculation implemented by COORDINSPECTOR follows the approach presented in the previous section, thus starting by labeling the vertices based on rules identifying communication primitives. At the moment of writing, COORDINSPECTOR is only instantiated with rules identifying web services communications, distinguishing between synchronous and asynchronous calls as well as between invocation and provisioning of functionality using web services. Other sets

The research reported in this paper is supported by FCT, under contract POSC/EIA/56646/2004, in the context of the IVY project.

[1]The tool is available from http://www.di.uminho.pt/∼nfr

[2]Source: http://en.wikipedia.org/wiki/CLI_Languages

[3]http://www.aisto.com/roeder/dotnet

of rules can, however, be easily added.

The graph pruning and slicing operations were once again implemented by following the specifications presented in the previous section and implemented by a series of graph traversal algorithms and transformation functions.

COORDINSPECTOR is also able to depict and navigate through both the calculated MSDG and CDG graphs, by resorting to the Microsoft Research GLEE graph library. The graphs provide different colors for the vertices, based on the labels the vertices hold, which facilitates direct manual reasonings over the graphs.

The graphical presentation of the graphs is also able to supply the user with specific vertex information, like labeling and the CIL code captured, by applying a double click on a particular vertex of the graphs.

Code generation in COORDINSPECTOR though based on function $\varphi$ defined above, was not implemented as a syntax oriented operation. Instead, this functionality is implemented by using and extending the same graph traversal operations that were defined for the labeling process of the MSDG.

## III. CONCLUSIONS AND FUTURE WORK

## REFERENCES

[1] F. Tip. A survey of program slicing techniques. *Journal of programming languages*, 3:121–189, 1995.