Automatic Dynamic Data Structures Recognition to Support the Migration of Applications to the Cloud

Beniamino Di Martino, Department of Industrial and Information Engineering, Second University of Naples, Aversa, Italy
Antonio Esposito, Department of Industrial and Information Engineering, Second University of Naples, Aversa, Italy

ABSTRACT

The work presented in this manuscript describes a methodology for the recognition of Dynamic Data structures, with a focus on Queues, Pipes and Lists. The recognition of such structures is used as a basis for the mapping of sequential code to Cloud Services, in order to support the semi-automatic restructuring of source software. The goal is to develop a complete methodology and a framework based on it to ease the efforts needed to port native applications to a Cloud Platform and simplify the relative complex processes. In order to achieve such an objective, the proposed technique exploits an intermediate representation of the code, consisting in parallel Skeletons and Cloud Patterns. Logical inference rules act on a knowledge base, built during the analysis of the source code, to guide the recognition and mapping processes. Both the inference rules and knowledge base are expressed in Prolog. A prototype tool for the automatic analysis of sequential source code and its mapping to a Cloud Pattern is also presented.

Keywords: Algorithmic Recognition, Cloud Computing, Dynamic Structures, Parallel Skeletons, Prolog

1. INTRODUCTION

During the past years the IT domain has been shaken by the rapid expansion of the Cloud Computing offer, with a growing number of Cloud providers literally flooding the market with new services, resources and platforms. This has also caused an important shift in the programming paradigm, since now most of the new applications are directly developed for and deployed on the Cloud, while existing software is slowly adapted and migrated to the new platforms. Despite its growth and widespread use, Cloud Computing suffers from a general lack of interoperability.

DOI: 10.4018/IJGHPC.2015070101
among different services and a low portability of applications across platforms. This is caused by the extreme variety of services offered by Cloud providers, often exposing non uniform and incompatible interfaces. In order to solve such issues, several standardization efforts have been carried out (CIMI (Davis and Pilz, 2012) and TOSCA (Binz et al., 2012) are remarkable examples). Also, catalogs of Cloud Patterns have been published, both by commercial vendors and independent researchers, to provide ready-to-use design solutions to most of the development and deployment issues which arise with Cloud Computing. One of the main drawbacks of such patterns is the lack of a formal machine-readable standard for their representation, as each catalog represents and describes them differently. In order to address such an issue, the authors have presented in (Di Martino et al., 2015) a semantic-based representation of both Cloud patterns and services to support the automatic composition and orchestration of Cloud applications. Methodologies and techniques to support and ease the (automatic) adaptation of existing software to new programming paradigms are highly desirable, as they would greatly reduce the time-consuming operations needed to actually perform the transformation of the code for the new environment, and they would also aid programmers in avoiding errors and bugs. In order to efficiently support the migration of existing applications to the Cloud, it is necessary to clearly identify the components which constitutes the original software and then map them to a set of suitable Cloud services and resources. The identification of the data structures used in the source software, together with the operations executed on them, can provide useful directions for the re-factoring of the applications and support developers in the complex process of migrating their existing code to a target Cloud platform. Several practical application are enabled by the recognition of data structures and the analysis of imperative sequential code. The automatic optimization of sequential code, in particular, can greatly benefit from algorithmic recognition: when specific parts of programs and portions of code are identified, they can be processed and parallelized, or method implementations can be substituted with calls to functions provided by optimized libraries.

Innovation and adaptation of existing software is not the only application of algorithmic recognition. Programmers often need to manage and maintain code which has been previously manipulated by several other developers: documentation regarding such manipulations can be out-dated or simply missing. So, the possibility to automatically analyze source code and retrieve information on its structure and exposed functionalities can represent a good means to re-build the missing documentation.

In the present paper a methodology and a set of logical rules, for the recognition of dynamic data structures from sequential imperative source code, are described. Also, a prototype tool which implements the presented technique and rules is presented. The presented research effort aims at expanding and refining the previous results in the same field, provided in (Di Martino, 1996) and (Di Martino, 2004). Exploiting such works as a baseline, the authors provide new data structure definitions and inference rules. By means of a compiler, built upon the Rose Compiler (Quinlan and Lao, 2011), the sequential source code of an application is automatically analyzed and information on its structure, components and dependencies is retrieved. Such information is then used to fill a knowledge base, which represents the starting point for a series of inferences and reasonings, enabled by a set of logical rules expressed in Prolog. In this way, it is possible to detect the presence of specific data structures and operations. Since the proposed methodology is not focused on supporting the re-factoring of sequential applications for just a single target environment, but is intended to be completely agnostic of the final deployment platform, it strongly relies on Skeletons, which can be used as a basis for later specific implementations. The mapping of the elements detected in the source code to specific Skeletons’ parts, represents a fundamental step of the migration process. Only after such a mapping, and according to the
Exploiting P2P Solutions in Telecommunication Service Delivery Platforms
www.igi-global.com/chapter/exploiting-p2p-solutions-telecommunication-service/40834?camid=4v1a

An Architectural Overview of the GReIC Data Access Service
www.igi-global.com/chapter/architectural-overview-grelc-data-access/20512?camid=4v1a