Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 24 (2008) 711–719
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

LGF: A flexible framework for exposing legacy codes as services☆

Bartosz Baliś*, Marian Bubak, Michal Wegiel

Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland

Received 16 February 2007; received in revised form 22 November 2007; accepted 17 December 2007
Available online 6 January 2008

Abstract

Scientific applications, usually written in Fortran or C, need to be adapted to work in modern environments for conducting scientific computations, which are based on web services or components. This work presents a Legacy to Grid framework (LGF) which enables semi-automatic virtualization of legacy codes as grid services. In contrast to existing work, LGF goes beyond a simple adapter-pattern approach; it follows a two-tier design in which the adaptation service layer is loosely coupled with the legacy back end layer. We present the architecture of our framework, its basic operation, and its capabilities: support for different interaction patterns with a legacy code and for computation migration to enable fault-tolerance and dynamic load balancing of legacy codes. We also present a case study of our framework's usage. We conclude with an evaluation of the impact of the two-layer architecture on the application's performance and the tradeoff between flexibility and performance by measuring the overhead of virtualization for different granularities of the legacy code.

Keywords: Legacy code; Grid services; Virtualization

1. Introduction

Legacy codes are those libraries, command line applications or other types of systems that were developed in technologies older than currently used in modern computing environments. An example is the current trend in development of systems for conducting e-Science experiments [7]. While the environments themselves are built as Grids based on web services or components following a kind of scientific workflow approach, the scientific applications which are supposed to run in those environments are often written in Fortran or C. Rewriting those applications from scratch or even refactoring them for new technologies would be either too expensive or impossible due to the lack of code documentation or specialists in programming in old technologies. Consequently, tools and frameworks to adapt legacy codes as services or components are created.

In this paper, we present a Legacy to Grid framework (LGF) whose goal is a semi-automatic adaptation of legacy codes to grid services, and to aid in the execution management of those codes. This work follows our previous effort [1,2]. We present a new design of the LGF framework followed by its operation scenarios. We also present considerations regarding advanced execution management of legacy codes supported in our system by means of dynamic load balancing and fault tolerance mechanisms. In the practical evaluation part of this paper, we present a case study to show the usage of the framework, its basic performance evaluation concerning overhead of the adaptation, as well as an analysis of the relationship of a legacy code's granularity to the adaptation overhead.

2. State of the art

This section presents a comprehensive overview of representative approaches to adapting legacy codes to modern environments. After presenting the approaches, we summarize their deficiencies.

A component wrapping approach to migration of legacy command-line applications to modern environments is presented in [6]. To create a wrapper, a specification is needed which includes the command that is used to run the application, pre- and postconditions (expressed as commands to be executed) which must be satisfied before and after the execution

☆ This work is partly supported by EU-IST Project CoreGrid IST-2002-004265, and EU-IST Project Virolab IST-027446.
* Corresponding author at: Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland.
E-mail addresses: balis@agh.edu.pl (B. Baliś), bubak@agh.edu.pl (M. Bubak), mwegiel@gmail.com (M. Wegiel).

0167-739X/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2007.12.001

of the legacy application, as well as a specification of the application's input and output. The paper also demonstrates how wrapped components can be composed in a distributed system via the JINI technology, using the ACME ADL to describe the composed application.

In [9], web service wrapping of command line legacy applications is presented. The work introduces a framework which can be used to instantiate legacy applications as services, invoke them in a secure way, and compose them into workflows. In order to invoke a legacy application, the value of input parameters, the location of input files, and the location to store outputs should be provided. This information must be contained in a Service Map Document which is loaded into the portal used to run a workflow.

The work presented by in [11] introduces the CAWOM tool and demonstrates its usability to wrap legacy command line applications as CORBA components. This approach is distinguished by the possibility to specify, using a formal language, the format of the legacy system's responses. This allows for more complex interactions with a legacy system, including synchronous and asynchronous calls, and ultimately a session-based interaction, for example, with a remote debugger. While this enables a great flexibility in terms of the interaction with the legacy system, it also complicates its adaptation. The tool's architecture distinguishes a server thread responsible for interaction with the client, and a mediator thread which interacts with the legacy system. Though this is hardly a two-layer architecture, this is a good step towards a flexible approach, fully exploited by LGF.

Grid Execution Management for Legacy Code Architecture (GEMLCA) [5] provides a general solution for deployment of legacy code applications as grid services without code re-engineering. The GEMLCA high-level architecture comprises: the *front-end layer* consisting of a set of grid services providing an access point for clients, the *internal layer* managing the legacy program environment and grid job behaviour, and the *back-end layer* virtualizing the grid infrastructure to facilitate system porting.

Another approach, presented in [8], introduces JACAW (JAva-C Automatic Wrapper) and MEDLI (MEdiation of Data and Legacy code Interface) — tools for semi-automatic conversion of legacy C/C++ interfaces to their Java equivalents. JACAW uses JNI in order to bridge between C and Java code. MEDLI allows for conversion of the Java code generated by JACAW to components that can be deployed using the Triana [10] workflow composer. This approach lacks the scalability and flexibility due to the tight coupling of legacy code with the service container.

Overall the mentioned approaches feature all or some of the following limitations:

- Practically all mentioned solutions focus on command line applications and follow a simple adapter pattern to automatically create legacy code wrappers based on the specifications of inputs and outputs of the target legacy application. Almost no solution supports fine-grain legacy code, such as libraries.
- In terms of the interaction with the legacy code, most of the approaches do not go far beyond what is available as a standard grid job submission mechanism (e.g. GEMLCA). The service front-end is in fact a service-oriented way to set up parameters, submit a job and retrieve results. Such a coarse-grain batch-oriented approach lacks the flexibility needed for automatic workflow discovery, does not support virtualization of legacy systems (e.g. databases), and prevents more interactive (e.g. session-based) interactions with a legacy code.
- In many cases, the legacy code is tightly coupled with service or component containers (e.g. JACAW/MEDLI). This limits the flexibility in deployment, execution and interaction with the legacy code.

3. Legacy to grid adaptation framework

After the analysis of the limitations of existing systems, we have adopted the following goals for a flexible solution for adapting legacy codes:

(1) Provide a framework that enables easy adaptation and deployment of legacy codes that goes beyond simple adapter-pattern wrapping, and supports both fine-grain and coarse-grain legacy codes.
(2) Support various interaction patterns with the legacy code, including synchronous, asynchronous, concurrent, transactional and session-based interactions.
(3) Enable mechanisms to manage the execution of the legacy code including the static and dynamic load balancing, as well as fault tolerance.

Based on those goals, the basic decisions that we have taken are as follows:

(1) The framework features a two-tier architecture which comprises the *adaptation and management layer* and the *legacy backend layer* which are *loosely coupled*.
(2) The legacy backend layer operates in a client fashion to the adaptation and management layer to enable easy deployment of the legacy code, migration of computation and advanced interaction patterns.

In Fig. 1 we present the architecture of the LGF framework. The system essentially encompasses three distributed components: a *Service Client*, a *Service Manager* which is in fact a set of web services (with the associated resources) deployed in a hosting environment, and a *Worker Job*. From the life cycle perspective, we distinguish between permanent and transient components. The former comprise a static part of the system. The latter represent dynamically created and destroyed entities whose lifetime spans its interaction with a single client.

Service client. The cooperation with legacy libraries is fully transparent, i.e. the clients are not aware of the underlying legacy code. We assume a thin-client approach so that mobile resource-constrained devices are not disqualified. Full isolation between concurrent clients is guaranteed. We do not impose any restrictions on the technology or language in which clients are developed — they are only required to support secure invocations of web service methods.