42nd international convention
May 20 - 24, 2019, Opatija, Croatia

Lampadem tradere

mipro - innovative promotional partnership

mipro proceedings
MIPRO 2019
42nd International Convention
May 20 – 24, 2019
Opatija, Croatia

Proceedings

Conferences:
Microelectronics, Electronics and Electronic Technology /MEET
Distributed Computing in Data Science and Biomedical Engineering /DC
Telecommunications & Information /CTI
Computers in Education /CE
Computers in Technical Systems /CTS
Intelligent Systems /CIS
Information Systems Security /ISS
Business Intelligence Systems /miproBIS
Digital Economy and Digital Society / DE-DS
Engineering Education /EE
Software and Systems Engineering /SSE
Composability, Comprehensibility and Correctness of Working Software /4COWS
MIPRO Junior - Student Papers /SP

Edited by:
Karolj Skala
Continuous Integration in Distributed Applied Software Packages

A.G. Feoktistov¹, S.A. Gorsky¹, I.A. Sidorov¹, A. Tchernykh²,³,⁴

¹ Matrosov Institute for System Dynamics and Control Theory of SB RAS, Irkutsk, Russia
² Computer Science Department, CICESE Research Center, Ensenada, B.C., Mexico
³ Ivannikov Institute for System Programming of RAS, Moscow, Russia
⁴ South Ural State University, Chelyabinsk, Russia

agf65@yandex.ru, gorskysergey@mail.ru, yvan.sidorov@gmail.com, chernykh@cicese.mx

Abstract – The paper addresses a new approach to supporting the continuous integration of modules for distributed applied software packages (scientific applications) that are developed in the Orlando Tools framework. We consider the packages for implementing experiments in heterogeneous distributed computing environments that integrate Grid systems, cloud platforms, and resources of public access supercomputer centers. Such packages are characterized by the modular structure, intense change of algorithmic knowledge implemented by modules, computations scalability, and execution in heterogeneous resources. In the packages, computational processes are represented by problem-solving schemes that closely correlate with the workflow concept. The essence and novelty of the proposed approach come from the merge of a methodology creating the packages with the modern practice of software development based on continuous integration. We automate and unify module assembly processes on both the dedicated servers and package developer machines through using the specialized virtual machines. In contrast to the well-known tools, we support the synthesis of the test problem-solving schemes in the automated mode. The practical applications of the developed prototype of the continuous integration system show the significant reducing the time of debugging, testing, and deploying the package modules.

Keywords - heterogeneous environment, scientific applications, modular approach, continuous integration, subject knowledge

I. INTRODUCTION

An applied software package (scientific application) is a set of applied programs (modules) designed to solve the defined class of problems in the specific subject domain. The package modules implement models and algorithms for solving these problems. In such a package, the computational process is described by a problem-solving scheme, which determines information and logical relations between modules. A particular case of the problem-solving scheme is the workflow scheme [1].

The progress of distributed computing technologies has led to significant changes in the architecture of applied software packages [2]. The architecture keeps the modular structure but becomes the distributed one oriented to a heterogeneous computing environment.

The hybrid models that combine cluster, grid and cloud computing in the unify environment provide the opportunity of sufficient increasing the efficiency and flexibility of the resource use. At the same time, it creates new challenges for the management of software packages in solving large-scale problems [3, 4]. These challenges are owing to the existing differences in the cloud and Grid computing models and the continued modifications of the software, hardware, and information resources of the environment [5-7]. The need for the package software evolution enhances these challenges.

Developers are working on the following problems:

- Reconfiguration of package environments,
- Modification of package software or developing new software taking into account a difference in cluster, grid and cloud resource characteristics,
- Supporting the correctness of interaction between different versions of software modules in problem-solving schemes,
- Determining the correct conditions of these versions,
- Integrating the changed sources of subject information with the data structures of packages.

Solving the aforementioned problems is related to applying methods and tools for continuous software integration [8].

The continuous integration support is a non-traditional capability of tools for the development of applied software packages including tools for creating scientific applications that are based on workflow [9].

Such tools are focused on the development of problem-solving schemes that have no changes or have minor changes. However, often, they are not ready to fully support the continuous integration combined with conceptual modeling that traditionally used in such applications.

It is related with the implementation complexity of obtaining and using the additional heterogeneous resources, complexity of structured knowledge about software and hardware infrastructure, and administrative policies in its nodes.
We propose a new approach to continuous integration of modules for distributed applied software packages and develop a special Orlando Tools framework [4]. In the proposed approach, we merge a methodology of creating distributed applied software packages with the modern practice based on continuous integration using subject domain knowledge. The main objective of our study is to ensure continuous integration for the packages whose domains are subject to intense changes in algorithmic knowledge used for solving problems.

The paper is structured as follows. The next section briefly reviews related works on the tools for continuous integration. Section 3 presents the architecture and main capabilities of the Orlando Tools framework used for the development of distributed applied software packages. The main aspects of the proposed approach to continuous integration in Orlando Tools are considered in Section 4. Section 5 presents the experimental analysis of applying the developed prototype of continuous integration. Section 6 highlights the conclusions of the paper.

II. RELATED WORK

The modern software development requires a variety of tools to increase its speed and quality and minimize the possible troubleshooting in the further applying the developed software [10]. Nowadays, there is a wide spectrum of universal tools supporting different functions of continuous software integration. Among them, for example, there are CircleCI [11], Jenkins [12], TeamCity [13], Travis [14], GitLab [15], systems described in [16, 17], etc. The main functions of continuous integration tools are receiving, storing, and testing software versions including the preparation and processing of the test data.

Well-known tools of continuous integration are CruiseControl [18] (no supported) and CircleCI. At the same time, some free tools have significant functional limitations (TeamCity) and do not provide user support with detailed documentation (Coverity and Hound CI [16]).

Notice, that many systems do not function under OS Windows (CircleCI and Buddy [19]), which very popular among developers and end-users of packages.

There are significant difficulties with combining tools for implementing different functions into a joint continuous integration environment due to the use of various models and formats of the data representation, and information and computational models that describe the processes of continuous integration (Travis [14] and BuildMaster [19]).

GitLab and Jenkins are the most popular systems for continuous integration, but they lack the conceptual modeling tools needed to support operation at the subject domain level. There is no well-known experience in the practical use of these systems related to distributed applied software packages. Based on the survey works [9, 20-22], we can also conclude that workflow management systems do not provide their users with advanced specialized tools of continuous integration.

Based on the analysis of the aforementioned tools, we conclude that there is a wide set of disjoint software that implements functions (or their restricted combinations) of continuous software integration. In this regard, the continuous integration tools that proposed in the paper and unified into the uniform technological sequence are aimed for a partial solving the above-listed problems.

III. DEVELOPMENT OF DISTRIBUTED APPLIED SOFTWARE PACKAGES

We developed the Orlando Tools framework to support large-scale scientific and applied researches [4]. This framework enables the development of distributed applied software packages and provides tools for the creating subject-oriented computing environments for executing packages [23]. Within such environments, various computing infrastructures that support both Grid and cloud computing are integrated.

In the paper, we represent a new subsystem of the Orlando Tools framework. This subsystem supports continuous integration of modules for distributed applied software packages.

We consider continuous integration in the context of developing a special class of the packages that provide the opportunity of the active evolution of algorithmic knowledge by developers and end-users.

The package modules can be developed and modified by different developers and end-users. The development and maintenance of each package consist in merging the actual versions of the modules into a general problem-solving scheme for nodes with different computational characteristics. This process can be performed from several times a month to several times a day.

Fig. 1 shows the updated Orlando Tools architecture that now includes the following main components:

- User interface to other Orlando Tools components (it implemented as a web application based on the JavaScript and PHP languages),
- Conceptual model designer that provides the capability for declarative description of the algorithmic knowledge, information about hardware and software infrastructure, and parameters of administrative policies in environment nodes,
- Subsystem for continuous integration of modules,
- Knowledge base that contains the conceptual model,
- Executive subsystem that includes the interpreter of problem-solving schemes, scheduler of computational processes, and generator of jobs,
- Computations database that keeps all information about computational experiments: the problem formulations, problem-solving schemes, and parameter values.

The conceptual model, its knowledge layers, main model objects, relations between them, and various ways of problem formulations using this model are described in [23, 24].
We also provide the special Orlando Tools server for creating the subject-oriented computing environment for the developed packages. It supports the user interface and executive subsystem. The Orlando Tools server is represented in the form of a configured virtual machine with the preinstalled software of our framework (the server is available by request to authors). In the future, we plan to use alternative server configuration tools (for example, Ansible [25]). The server enables to include to the package environment the following components that differ in their software and hardware characteristics:

- High-performance computing clusters with the local job management systems (SLURM, HTCondor or PBS Torque),
- Computational nodes with the non-portable software under the OS Linux management,
- Clusters of dedicated resources (cloud) represented by the special Orlando Tools virtual machines, which host the package modules and software to manage their execution, personal computers, and high-performance servers,
- Remote computational and information resources available through the Grid or web-service API.

Challenges in the development and use of distributed applied software packages occur equally for both the developers and end-users. Developers and environment administrators should properly configure the environment for the reliable and high-performance package operations taking into account subject domain specifics for solved problems. They place the package components and external system software into the environment nodes and set their configuration parameters. When a package and other systems are configured, a significant difficulty for the end-users consists in the selection of a high number of parameters at the problem-solving process implementation (algorithms, values of their control parameters, compilers, input and test data, problem-solving schemes, target nodes, etc.) to optimize the distributed computing.

We add the new subsystem for continuous integration of distributed applied software packages to partially mitigate the aforementioned challenges. Through the special API, it provides interaction with external tools of continuous integration (see the right side of Figure 1), including source repositories, compilers, composers, program libraries managers, etc. This subsystem elicits the necessary information about the specifics of modules and environment nodes, in which these modules are executed, from the knowledge base and computations database.

IV. CONTINUOUS INTEGRATION WITH ORLANDO TOOLS

Generally, the process of developing applied software using Orlando Tools involves the following stages:

- Structural analysis of the package subject domain and designing the conceptual model including the specification of parameters, operations, and modules for this domain,
- Development and modification of the program code for modules that implement operations,
- Compiling, debugging and testing modules in the heterogeneous distributed computing environment,
- Placing compiled versions of modules in repositories or installing modules on resources,
- Modification of the conceptual model,
- Debugging and testing problem-solving schemes with the new or modified versions of modules.

Each stage requires efforts of the package developer. The coordination of different stages and support of informational and logical relations between the tools entail additional overhead costs. Multiple repetitions of these stages actualize tools to automate the process of continuous integration of modules of distributed applied software packages. We propose the following scheme of...
supporting continuous integration of the package modules in Orlando Tools: Development and modification, Automatic installation and updating, Synthesis of test problem-solving schemes, Preparing the debugging and testing, and Debugging and testing.

Development and modification of the source code of the modules, compilation, debugging and testing are carried out automatically by Orlando Tools when changes related to the package modules are committed in the Git system [26]. Automatic installation and updating the modules on resources are implemented by the Orlando Tools executive subsystem. They are carried before or during the computational experiment depending on the end-user preferences. For the synthesis of test problem-solving schemes, we use the non-procedural formulation: Calculate Y knowing X, where X and Y are sets of parameters of the package subject domain. Selection of the modules to be included into schemes is implemented through wave methods of the computations planning over the conceptual model. It specifies the information and logical relations between modules in term their input and output parameters. Preparing the debugging and testing are based on the execution results obtained earlier and stored in the computations databases. During these processes, Orlando Tools uses data from completed experiments. Debugging and testing problem-solving schemes are complemented by additional operations to control the correctness of the modules execution. The data flow diagram of the Orlando Tools continuous integration process showed on the Fig. 2.

We apply the tools of the Git system for version control of the module source code of packages. [26]. It supports the rapid separating and merging versions of the module source code and includes visualization and navigation tools in the software development process, taking into account its non-linear history.

In Orlando Tools, we use the GitLab Self-Managed platform for the repository management. GitLab is the popular platform, which provides programming code analysis, error tracking, module versions testing, etc. In addition, GitLab provides advanced tools to manage modules testing within continuous integration. We implement module assembly processes both on the dedicated servers and on package developer machines using the pre-installed GitLab Runner agent.

We apply Conan.io to manage the dependencies of the libraries that are used in the source code of the modules [27]. Conan.io is a portable package manager intended for C and C++ developers. In addition, it enables developers to manage builds from sources, dependencies, and precompiled binaries for any language.

We apply the JFrog Artifactory system to operate with binary versions of modules [28]. It provides the needed interfaces for interacting with various package managers (for example, Git LFS Registry [29], Docker Registry [30], and Composer [31]).

We develop the prototype of a system that realized the proposed scheme of supporting continuous integration, delivery, and deployment of the package modules in Orlando Tools. The interaction of Orlando Tools with the above-listed systems is implemented using special API. A comparative analysis of the main capabilities of GitLab and the developed prototype shows that last one provides new additional opportunities related to the features of the package's development are provided (Table I).

<table>
<thead>
<tr>
<th>TABLE I.</th>
<th>CAPABILITIES COMPARISON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability</td>
<td>GitLab</td>
</tr>
<tr>
<td>Continuous integration of modules</td>
<td>+</td>
</tr>
<tr>
<td>Delivery of modules</td>
<td>+</td>
</tr>
<tr>
<td>Deployment of modules</td>
<td>+</td>
</tr>
<tr>
<td>Automation of synthezing test problem-solving schemes</td>
<td>–</td>
</tr>
<tr>
<td>Automation of the through continuous integration, delivery, and deployment that are applied to both the modules and schemes</td>
<td>–</td>
</tr>
</tbody>
</table>

V. EXPERIMENTAL ANALYSIS

We apply the prototype of the system for continuous integration of modules to develop the package of global minima search of multi-extremal functions [32]. The package includes modules that implement various modifications of the multi-start algorithm in the context of different multi-extremal functions (Griewank, Rastrigin, and other test functions [33]).

Fig. 3 shows one of the schemes for solving the optimization problem described above. It includes four operations. SYS_GetRes supports the resource allocation for the problem-solving scheme execution. Generate generates a set of initial points for the multi-start algorithm. Gradient carries out descent from the initial
point into a local minimum point by the gradient method. Complete determines the global minimum. The operations description is provided in [32].

In the package, each of these operations can be implemented by different modules. They depend on the methods of forming the set of initial points in the multi-start algorithm, allocation of resources for problem-solving, distribution of a computational load between resources, and multi-extremal functions used to test modifications of the algorithm.

At the same time, a variety of problem-solving schemes is due to the variety of operations. New problem-solving scheme for some algorithm modification can be created by excluding some operations from the existing scheme and adding other operations of the package. Depending on the algorithm modification, problem-solving schemes include from 3 to 5 operations.

Fig. 4 demonstrates the average time that a developer spent on the modifying one module without continuous integration, using automation of continuous integration in Orlando Tools, and applying the external tools of continuous integration (GitLab only) in the manual use manner. These results were obtained on the base of average operation time evaluations provided by the participants of two developer groups those developing multi-start algorithm versions for the same set of multi-extremal functions in above-listed modes. In a one week, there were 31 and 37 module modifications in the first and second groups correspondingly.

In the first case, we see the large time that developer spends at each stage of the module modification.

In two other cases, we see the significant reduction of developer efforts. However, in comparison with continuous integration in GitLab only, such process in Orlando Tools enables the developer to decrease the time at the stages of the module modification and verification of the results. A developer does not spend significant time at the five stages due to their full automation in the Orlando Tools continuous integration.

When the package software modification is needed (Fig. 5), Orlando Tools demonstrates obvious advantages at all stages in comparison with the other two ways of continuous integration. In Orlando Tools, two stages of continuous integration are fully automated. In Fig. 5, we used average user’s evaluations of the operation time for typical processes in the Orlando Tools.

In practice, the development and modification of packages for the heterogeneous distributed computing environment can cause additional labor costs. Especially, it is significant with the manual use of external tools of continuous integration and human factors, for instance, various errors from the developers and end-users. Automation of the technological sequence of continuous integration significantly decreases the probability of errors and reduces the labor costs to correct them.

In the paper, we first time evaluated the effectiveness of the developed prototype system for continuous integration of modules. All experiments are carried out in the environment based on the heterogeneous resources of the Matrosov Institute for System Dynamics and Control Theory of SB RAS and public access computer center “Irkutsk Supercomputer Center of SB RAS” [34].

VI. CONCLUSION

In the paper, we propose a new approach to the continuous integration of modules for distributed applied software packages. To this end, we develop a special Orlando Tools framework.

A key feature of such packages is a possibility of the change of algorithmic knowledge implemented by modules that can be developed and modified by different developers and end-users.

We propose the scheme of the automation and unification of module assembly processes applying external continuous integration tools unified through the Orlando Tools server.

In this scheme, we unify the external tools into the uniform technological sequence. We use the conceptual model for specifying information and logical relations between the external tools and packages.

Orlando Tools automates the launch of the required external tools and provides data transfer between them. In contrast to the well-known tools of similar purpose, it also supports synthesizing test problem-solving schemes in the automated mode.

We develop the prototype system for continuous integration of modules for distributed applied software packages. The practical experiments show the significant reducing in the time of debugging, testing, and deploying the package modules compared with traditional approaches.
Nevertheless, further study is required. Information elicited from subject data and results of solving problems obtained in scientific experiments are often heterogeneous and subject to frequent changes. In this regard, a flexible, knowledge-based model is needed to set relations between primary information and data structures used in the packages. In addition, we plan to use the test results for predicting module execution time when Orlando Tools allocates these resources. Such a prediction allows increasing the resource utilization and reducing the execution time of problem-solving schemes. These studies will be the subject of future work.

ACKNOWLEDGMENT

The study is supported by the Russian Foundation of Basic Research, projects no. 19-07-00097 and no. 18-07-01224. This work was also supported in part by Basic Research Program of SB RAS, project no. IV.38.1.1.

REFERENCES


