International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 3 Issue: 6 4214 – 4217

Build Automation on an Order Management System

Shakti. S. Putra
Depatment of Computer Science and Engineering
M.S. Ramaiah Institute of Technology
Bangalore- 560054, India
shaktisputra@gmail.com

Ms. Rekha Nair
IBM India Pvt. Ltd.
Embassy Links,
Bangalore-560071, India
reksnair@in.ibm.com

Prof. Sanjeetha R
Department of Computer Science and Engineering
M. S. Ramaiah Institute of Technology
Bangalore 560054, India
sanjeetha.r@msrit.edu

Abstract—One of the common problems faced in companies is the maintenance of multiple products and multiple versions of each product that are used at the same time. Engineers who do this have to manually run commands to check the product and its version, then identify the software to be installed, test fixes, and then continue with other maintenance processes.

In this paper we aim to automate this process by designing a polling system that continuously checks for fixes available for a particular product and maintains information about the same.

Keywords*-build automation; test automation*

****

I. INTRODUCTION

An Order Management System is a system that is used for order entry and processing. Multiple large-scale enterprise applications that provide capabilities of order management are available in the market. Bugs found in such large scale applications, after the application has been released are fixed by Fix Packs (FP). A Fix Pack is a software maintenance package that is used to fix one or more software defects.

To test FPs, the runtimes associated with products must be kept updated. This paper proposes a system that aims to automate the process involved in testing these FPs using build automation.

Build automation is the process of automating or scripting tasks such as compiling source code into binary code, packaging binary code etc.

II. PROPOSED SYSTEM

Enterprise applications like the Order Management System have multiple products and multiple versions of each product being used simultaneously. Each of these products has a runtime associated with it. The testing of fixes requires the product runtime to be maintained at the latest drop (a version of a fix that is used to test the fix before the fix pack is released).

This process involves identifying available drops, validating drops and invoking the installation of the required drop for a particular runtime. Currently, the person testing a fix pack must check the drop folder for new drops, decide which drop to install based on the drop installed in the runtime and then invoke the install, build enterprise archive (EAR) and backup EAR commands. This process requires a lot of time and effort if performed manually. Hence, in this paper we propose a system that automates the above mentioned process using build automation.

The proposed system would perform three main functions: (i) identify and validate new drops, (ii) decide the next drop to be installed on a runtime, (iii) invoke the required installations

III. ARCHITECTURE

The system proposed in this paper solves the above mentioned problem through 3 main sub-systems: the drop poller, the install decision maker and the installer. The Controller would be used to control and co-ordinate the sub-systems. The Configurator would be used to configure the system and initialize product instances to be considered by the system.

A. Controller

The controller is responsible for co-ordination among the sub-systems. The controller is the first part of the system that starts when the system starts and runs on the main thread. The controller calls the configurator which configures the system. Based on the product instances specified in the configuration file, the controller then spawns new threads for each of the product instances for the purpose of polling and performing installations. The controller invokes the install decision maker after it checks if an installation can be performed on the runtime. The controller also verifies that no more than the maximum allowed number of installations are invoked on a runtime. The maximum allowed number of simultaneous installations can be set in the configuration file depending upon the system's capacity. The controller avoids unnecessary calls to the decision maker by performing the above mentioned check first.

B. Configurator

The configurator is used to set the properties of the system by reading them from a configuration file. The location of configuration file can be passed as an argument to the program, or can be set as a system property, or can be saved in a predefined location specified in the program. The configurator is used to initialize:

- The system properties
- The product instances
- Event handlers for each event
- Event handler properties

The configurator runs on the main thread and is the first part of the system that executes on start up. After the configurator finishes execution, control is returned back to the controller

C. Poller

The controller creates a new poller thread for each product instance specified in the configuration file. The poller performs three main functions:

- Identifies new drops in the drop directory
- Validates new drops
- Maintains a cached list of valid drops

The first function of the poller is to poll the drop directory (the directory where drops are stored) for new drops. If a drop is found, the poller performs a series of validations on the drop. The validations include checks to verify if the drop has been copied completely, if all the fields describing the drop match the drop name, if the drop is not on hold etc. If a drop fails a check, a suitable event is raised to indicate the occurrence of the error. Once the error is resolved, the drop is identified in the next scan and is added to the cache list.

The poller also continuously runs checks on cached drops and raised suitable events if a drop becomes invalid after it has been cached. This ensures that drops in the list are valid.

D. Installer/ Runtime module

The installer is the part of the system that invokes the commands needed to install the drops on a runtime. The installer module is present within the runtime module. The runtime module adds tasks to the task list, decides the task to be performed and performs the required tasks.

The controller starts a new runtime thread for each product instance specified in the configuration file. The runtime thread, checks the task list file for tasks. If a task is found, the runtime thread performs the task. Once the task is completed, it is removed from the task list. A task maybe one of three types: an install task, a build EAR task or a Backup EAR task.

When the install decision maker identifies a drop to be installed on a runtime, the controller raises an install request that is received by the runtime module. If the install request can be accepted, the runtime module adds the tasks to the task list for that runtime. If an installation is already in progress, the install request is not accepted by the runtime, and a negative response is returned to the controller.

If the task to be performed is an install task, the installer first checks if the task was started earlier. If yes, the exit status of the task is retrieved and suitable action is taken. If not, the required command is invoked and that runtime thread waits until the installation is completed. The exit status is checked and the installation log file is scanned for errors in the installation. If the installation was successful, the installer proceeds with the next task. If the installation failed, a suitable error is raised and the task list is cleared. The drop is set on hold indicating an error in the drop which prevent the same drop being picked up for installation again by the install decision maker.

E. Event Handler

The event handler is responsible for handling event raised by the system. Event handlers can be added to the system by implementing an interface provided. The event handler for each event can be configured and each event maybe associated with more than one event handlers. Each event handler can further be configured for each event. For example, an event maybe associated with a file dump handler and an email handler. The file dump handler can be further configured to write different events to different files. The email handler can be configured to send mails about different events to different ids.

IV. ALGORITHM

The system spawns new threads for each of the above mentioned sub-systems. In this section, we first describe the algorithm followed by the entire system and then explain the algorithms followed by the sub-systems.

A. Main Flow

The controller is the first part of the system that is executed on startup. The controller calls the configurator which is used to configure the system properties, initialize product instances and configure event handlers. The Configurator reads properties from a configuration file and sets the above mentioned properties.

The controller then creates new poller threads for each product instance that is initialized by the configurator. A new thread is created only for unique product instances. Two threads are never spawned for the same product instance.

New runtime threads are created by the controller for each product instance initialized by the configurator. The runtime thread is responsible for performing tasks on the runtime. The installations are performed on this thread.

The controller monitors the number of installations running at a time. If the maximum allowed number of installations are running, it prevents the install decision maker from being called until one of the installations ends. The controller maintains a list of installations in progress and validates this list periodically.

The controller also invokes the install decision maker that decides the next drop to be installed on a runtime. Based on the value returned by the decision maker, the controller sends an install request to the runtime.

B. Polling

The polling for drops takes place on a separate thread and is performed separately for each unique combination of product and version specified in the configurator.

The poller thread first validates the cached list of drops by performing a series of checks on each drop. If a drop fails an of the checks, a suitable event is raised and the drop is removed from the list.

The next step in the poller flow is validating new drops. The drop directory is scanned for drops that are not found in the cached list. Each new drop found in the drop directory is made to pass through a series of checks. If any check fails, a

suitable event is raised and the drop is not added to the list. Depending upon the check failed, the drop maybe put on hold.

This algorithm is used to maintain a list of available valid drops. This list is used by the decision maker to decide on drops to be installed.

C. *Runtime*

The runtime thread is responsible for maintaining a task list for a given runtime and performing tasks for that runtime. The thread first checks the task list for tasks. If a task is found, the corresponding method is called to perform the task. A task maybe an install task or a build EAR task or a backup EAR task. Once a task is completed, it is removed from the task list if the task was successful. If a task fails, all tasks from the task list are removed and a suitable event is raised.

The runtime thread keeps running until all tasks in the task list are completed and sleeps once the task list becomes empty.

The runtime thread also receives install requests from the controller. If an install request is received when an installation is in progress, that task is not added to the task list. If the request can be accepted, three tasks are added to the task list.

The tasks contain all the information required to perform tasks. This information is retrieved from the task list by the methods performing the tasks.

D. *Install Decision Maker*

The install decision maker is a critical part of the system that decides the drop that must be installed on a runtime. The controller invokes the decision maker. The decision maker retrieves the FP number, the drop number and the build number of the drop currently installed in the runtime. It then obtains the rule configured for the runtime. Based on the rule, it identifies the drop to be installed from the cached list. If no drop is to be installed for a runtime, the decision maker returns null. If the controller receives null, it does not invoke an install request. If the decision maker identifies a drop to be installed, it creates a drop entity object and returns the object.

The decision maker also handles situations where the drop to be installed is put on hold. In such cases, the decision maker checks for an earlier drop that has not been put on hold and is later than the drop currently installed. If such a drop exists, it is returned. If such a drop does not exist, null is returned.

E. *Event Handler*

The event handler uses the concept of reflections to handle events. Based on the handler configured for an event, the required handlers are instantiated dynamically.

Handlers can be added to the system by implementing specific methods.

The handlers for each event must be specified in the configuration file. A default handler can be configured which will be used if no handler is specified for an event.

The event handler also maintains a list of events triggered within an event window. The duration of the event window can be configured using the configuration file. If an event identical to one present in the list is raised, its handler is not called. This prevents identical events being raised repeatedly in short intervals of time.

V. CONCLUSION

In this paper we propose a system that automates the build process involved in testing of fixes for an enterprise application - the Order Management System. The system is robust and recovers from most errors automatically without the need for human intervention. The system uses multi-threading, and thus, individual components of the system work simultaneously without affecting the performance of other components of the system.

The challenge involved in developing an automation system is the need for minimal requirement of human intervention in case of failures. This challenge is overcome by the system by providing suitable error recovery methods for most commonly encountered error conditions.

ACKNOWLEDGMENT

We would like to thank, the Head of the Department of Computer Science and Engineering, M. S. Ramaiah Institute of Technology for his continuous support.

We would also like to thank our beloved principal and the management of M.S. Ramaiah Institute of Technology for their support and encouragement.

REFERENCES

[1] Mark N. Frolick Brian D. Janz, Cycle Time Reduction in Software Testing, Information Technology: New Generations (ITNG), 2012 Ninth International Conference on 01/2012

[2] Design Patterns, http://www.dofactory.com/net/design-patterns

[3] Yijun Yu, Reducing Build Time Through Pre compilations for Evolving Large Software, Uni Harrold College of Computing Georgia Institute of Technology Atlanta, GA

[4] Continuous Delivery, https://buildrelease.googlecode.com/hg-history/1998cd1d530b35b79740d7bf93f8915548136c25/Trunk/BreBooks/Continuous%2520Delivery.pdf

[5] Jenkins Continuous Build System, 2012, http://www.cs.colorado.edu/~kena/classes/5828/s12/presentation-materials/bowesjesse.pdf

[6] Manoranjitham, R, Jagdale, B.N.,Designing an automation framework to improve the performance of SIP based test suites PBX feature testing, 2014 IEEE Global Conference onWireless Computing and Networking (GCWCN), pp. 258-262, December 22-24, 2014

[7] Harrold M.J.,Reduce, Reuse, Recycle, Recover: Techniques for Improved Regression Testing,, IEEE International Conference on Software Maintenance, 2009. ICSM 2009.

[8] Buildbot http://buildbot.net/

[9] Apache Anthttp://ant.apache.org/

[10] An Introduction to Apache Ant, www.csee.umbc.edu/courses/undergraduate/341/.../Ant/intro-to-ant.ppt

[11] Robert C. Martin,The Principles, Patterns, and Practices of Agile Software Development; Prentice Hall, 2002.

[12] E. Gamma et al.,Design Patterns: Elements of Reusable Object-Oriented Software; Addison-Wesley, 1995.

[13] Singleton and Monostate Design Pattern, http://www.objectmentor.com/resources/articles/SingletonAndMonostate.pdf

[14] Rankin, C.,The Software Testing Automation Framework, IBM Server Group, 11401 Burnet Road, Austin, Texas 78758, USA

[15] Eun Ha Kim, Jong Chae Na, Implementing an Effective Automation Framework; in 33rd Annual IEEE International Computer Software and Applications Conference, 2009. COMPSAC '09. (Volume:2), pp. 534-538

[16] Software Testing Automation Framework, http://perl.org.il/presentations/software%20testing%20automation%20framework%20pii.pdf

[17] Software Testing Automation Framework, http://staf.sourceforge.net/

[18] Dan Ye,Automated Testing Framework for ODBC Driver,Nanjing, China.

[19] M. Kelly,Choosing a Test Automation Framework; 2003. http://www.ibm.com/developerworks/rational/library/591.html

[20] The Automated Testing Handbook,http://www.softwaretestpro.com/itemassets/4772/automatedtestinghandbook.pdf

[21] Test Automation and Software Development, http://www.tcs.com/SiteCollectionDocuments/White%20Papers/Test%20Automation%20and%20Software%20Development.pdf

[22] Order Management,http://searchfinancialapplications.techtarget.com/definition/order-management

[23] Order Fulfilment,http://www.businessdictionary.com/definition/order-fulfillment.html

[24] Managing Multi-channel Orders, http://ordercore.blogspot.in/

[25] Process Builder, http://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html

[26] Java Reflection, http://wwwusers.di.uniroma1.it/˜parisi/Risorse/java-reflection-explained-simply-manual-8up.pdf

[27] Java Theory and Practice: Decorating with Dynamic proxies, http://www.ibm.com/developerworks/library/j-jtp08305/

[28] Dynamic Proxies in Java, http://userpages.umbc.edu/˜tarr/dp/lectures/DynProxies-2pp.pdf

[29] Parallel Computing at a Glance, http://www.buyya.com/microkernel/chap1.pdf

[30] Parallelism, http://www1.ece.neu.edu/˜dschaa/docs/parallelism.html

[31] Multiprocessors and Thread Level Parallelism, http://www.cse.chalmers.se/edu/year/2009/course/DAT105/lectures/L5.pdf