6th International Conference on Applied Informatics
Eger, Hungary, January 27–31, 2004.

Objects management in Real-time in Java systems

Imre Fazekas

Department of Information Technologies,
University of Debrecen
email: ifazekas@inf.unideb.hu

Abstract

Information systems working continuously need a technology to ensure the possibility for software development in real-time. An Object Oriented (OO) system is a set of objects, which can be replaced, modified; so the behavior of objects will be changed, and we want to ease the programming work on a running Java software. For this purpose, we introduce Fluxion and CIA technologies.

On the basis of the analysis on the behavior of the previous software components, one can make structural or behavioral corrections without the decrease of performance in the new system. One can use the tools Agents and Wrappers [1-9] introduced by Odell et al. In such way, one can control any kind of operation on the same level of performance. An Agent [1] "slopes around" in the software, facilitating the debug, the control, the corrections of the process of operation of the software.

Keywords: Definition and manipulation of classes, Interfaces, Methods and properties in real-time; "Hot-swapping"; Replacement of objects; Modification of behavior; Software management

1. Introduction

"Software hot-swap" [9-17]: in the course of the working of an information system, one can replace components in the software, can make corrections on the basis of the analysis of the software.

In the last 3-5 years the following problem aroses: develop a software in real-time for the continuous work. It needs a new technology built upon the existing OO technology. It is known, that the OO environment is defined as a set of objects which are related and communicate with each other. In this paper we shall use the terminology of Gamma et al [17]. So, in the process of developing the software, we manipulate the behavior of certain objects, instantiate new objects, etc. The software "hot-swapping" or developing in real-time means replacement of objects. For this, we introduce the necessery basic tools:

- Dynamic classloading
- Definition and modification of classes and interfaces in real-time
- Modification of the behavior and the states of objects in real-time
- Performance improvement

We introduce two technologies the Fluxion and the CIA, which were implemented in Java.

1. Fluxion is a technology for realizing the "hot-swapping" in Java systems.

To most difficult problem is to preserve the stability of the system and the states of the objects placed in the memory during the development process. Transactions are acceptable with a minimal delay. The Fluxion gives the possibility of defining and installing classes, interfaces; manipulate objects in real-time.

2. Central Industry of Agents (CIA) is a technology for creating Agents and Wrappers in Java systems CIA gives the possibility of analysing software components, to perform some corrections respecting the behavior of objects while the software is running without performance decrease. This Fluxion-based technology allows us to define IT-Agents and Wrappers [1] this way we can build them into a system and perform the actions defined by us. There is no need for activities of programmers or knowledge of certain Application Programming Interfaces (APIs). In our hopes, one of the built-in Agent or user defined Agent "slopes around" in the Java software, facilitates the debugging, the control, and corrects operations and states.

In Section 2, we will define and describe the essence of our technology, the realization of the Fluxion and CIA systems. In Section 3, we check and verify the real performance of this technologies.

2. Realization

2.1. Fluxion

In the realization of Fluxion system we used a Java-based technology.

The features of Fluxion:

- developing in real-time,
- defining and modifing classes, methods, properties in real-time,
- management of persistent objects,
- assure the consistent running including the advance of performance

In the JVM[20] a class is identified by its name and the instance of the classloader. This way, one can make several registries to the same class using numerous classloaders. By using a custom classloader, we have a factory to manage classloading, class versions, instantiating, and secure distribution of system resources. In other way, every object creation must be performed through this classloader, so modification of the source is needed. The Fluxion Integrator contained by the Fluxion package, automatically transforms the source by rewriting all instantiation and object creation statements.

Managing object versions

By defining a new version of a class, we have to replace all its instances, without stopping the program or some of its parts. In a running OO application, the objects communicate with each other, so they are in relation. At direct relation, we can not make any changes without any side-effect, in other words any modification invokes a never-ending modification chain by taking effects on the caller cascading the whole calling chain. The direct relation is the common problem, i.e. we have to resolve it. Using the Dynamic Proxy Classes[21] technology and interface-based design, a custom utility can be defined to permit of use indirect relation between objects. In an indirect relation the caller references to an object, but the reference comes from the Fluxion system, and it relates to a wrapper object, which handles every invocation and access and may delegates it to a real object invoked by the caller. So, the real-time change does not diffuse through the call-chain, no one gets knowledge of it.

To support persistence handling, the Fluxion system uses an interface to define methods to ground for perform data-saving, and information preserving.

Changing objects in real-time goes hand in hand with resource allocation. The Fluxion system uses a custom object management utility to balance performance, and optimize garbage collection.

2.2. CIA

CIA, which realises a system with the facility of creating and controlling agents achieves real-time profiling and correcting. Framework for defining and implanting IT-Agents in an OO system without any precognition. Programmers can use the built-in Agents or define their own ones. This technology supports the following functions:

- easy-to-use User Interface,
- defining, coordinating, controlling Agents and Wrappers in real-time,
- consistent operation minimizing the decrease of performance,
- full control and supervision over the agents,
- creating and drawing measurments statistics.

We have defined the following built-in Agents in CIA.

- Measurement: measures the execution of defferent methods.
- Narrow: identifies the bottle necks in the system.
- Corrector: develops a software component [16,18] to insure reusability, and performance.
- Partial: it makes partial computing to identify common case, changes the structure of a software component for the better performance [18].

We want to manage Java systems based on Agents, in the simplest way if it is possible. Basically, a "creature" must be integrated to the running Java system. This creature has to perform actions, like measuring, correction or analysing. Our theoretical system based on the ASIG [9], creates the basis of an agent-based software management.

This creature, which is called an Agent, is able to perform any task defined by us irrespectively of the Java software structure. On the basis of ASIG, CIA must provide an agent system with the following properties:

- autonomic,
- interactive,
- intelligent,
- wrapper,
- cooperative.

For example, if we want to identify the bottlenecks in an OO software environment, we have to define an agent who adapt himself to the actual software, able to interact, cooperate with other agents to have some inner activities performed. Agents provide absolute capability to profiling, analysing, developing the software.

We prescribe important criterias:

- user activities must be performed in real-time,
- delay of the software component must be minimal,
- insure the freedom of development,
- realizable usage for defining, creating, debugging and controlling agents.

Realizing agents in a dynamic environment

In an OO system, an Agent is an object too, whose function is to integrate itself to the software and "slopes around" among objects, without so much as the system catch sight of it. In fact, an object is a code in the storage. In this way, an

Agent is a code fragment, that appears in the text of the code. The purpose of the Agent is to hack-in to the software, identify and find the code, rewrite it, collect information or anything it wants. After terminating its activities, it rewrites the code to the original state and disappears.

To coordinate lives and functions of Agents we have to have full control over them. In the CIA, an interface defines the abstract actions of an agent:

- start,
- stop,
- context registration [12-15].

The context insures the independence among objects, provides information to agents, gives us control over agents, the interactions of agents. It gives the agents the possibility of controlling an other agent, share information among each other. So, the context has an eye to everything. An agent has to be independent from the agent system, from the target OO system, and from other agents too. Therefore, the usage of Context is obligate.

In fact, an Agent is related to the context. It registers the result of agents' actions, perform all operation required by an agent.

To realize the interactivity of the user, i.e. starting, stopping and defining agents an easy-to-use user interface is needed. The context is a control point, it distributes information about activities, functionalities, states of the agents. It is the connection between the user and the agents. On the one hand, it provides services to agents, on the other hand, it provides services to the user to help creating, controlling agents.

Terminating Agents

There are three ways to terminate an agent.

- Specifing when it has to stop.
- Making an intervention to arrest it.
- The agent reaches a decision to stop itself.

The third case is the most interesting. On the basis of collected information from the running software, agent can reach a decision to stop itself. In this case, we should provide him with the necessary knowledge. In fact, the programmer defines when it "thinks" on stopping.

Solutions of self-controlled terminating:

- elapsing a given period,
- supervention of a certain event,
- fulfilling a given condition,

- arbitrary combination of the preceding methods.

The self-controlled terminating is useful when the condition of termination is defined by a software element. For example: number of calling a method, waiting for a defined value or state)

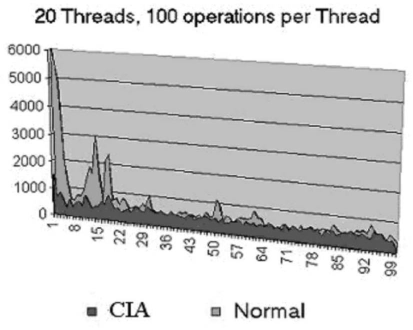
CIA based on Fluxion, performs all source modifications automatically, including:

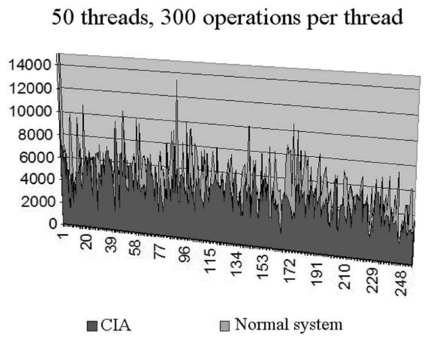
- building-in custom classloder,
- inserting invocation wrappers,
- generating agent context and agent factory,
- etc.

Running a CIA based application, one can define, instantiate or use a built-in agent, perform custom operations, harmonise works of agents, and so on.

3. Performance

The side-effects of Fluxion and CIA systems were tested on the Pegasi Web Server, the test-winner Java-based Web Server [10,11,19]. The two diagrams in Figure 1 demonstrate the performance of the two Pegasi systems, whose structure are roughly equivalent with the difference that one of them uses CIA technology, so a performance-measurer Agent is "sloping around" in it and travels from an object to an other while measuring its serving method. The horizontal axis represents the operations, the vertically represents the response time.





Giving requested information to the client threads, the CIA server is a little bit faster. The reason of this speed increase is the effective object creation by Fluxion. The dispersion of the delay of a response is smoother which results in a more stable load.

4. Conclusion

In fact, using Fluxion, we have the possibility to develop in real-time, to make any behavioral or structural change in the system increasing the performance of the original system. CIA based on Fluxion, allows to define IT-Agents to measure, correct the system or perform custom operations. With these two technologies, the software development in real-time in Java is resolved in full details.

References

[1] James Odell, David Levine 2OO0. Agent technology

[2] James Odell, 1999 Introduction to Agents.

[3] Bradshaw, J. (ed.). 1997a. Software Agents. American Association for Artificial Intelligence/MIT Press.

[4] Bradshaw, J. 1997b. An Introduction to Software Agent. In (Bradshaw 1997a).

[5] [Cag97] Caglayan, Alper, and Colin Harrison, Agent Sourcebook, John Wiley & Sons, New York, 1997.

[6] [FIPA] http://www.fipa.org

[7] [M99] Frank Manola, "Providing Systemic Properties (Ilities) and Quality of Service in Component-Based Systems." 1999. URL: http://www.objs.com/aits/9901-iquos.html.

[8] [MT99] Frank Manola and Craig Thompson, "Characterizing the Agent Grid." 1999. URL: http://www.objs.com/agility/tech-reports/990623-characterizing-the-Agens- grid.html.

[9] OMG Agent SIG. http://www.objs.com/ag'ent/

[10] Dr Job's Journal, www.dii.com

[11] Object Oriented Programming www.oop.com

[12] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-Oriented Software Architecture - A System of Patterns. Wiley and Sons, 1996.

[13] Automatic code generation from design patterns

[14] P. Coad. Object-Oriented Patterns. Communications of the ACM, V 35 N 9, Sept 1992, pp. 152-159.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.

[16] Schmidt, Douglas. Using Design Patterns to Develop Reusable Object-Oriented Communication Software, CACM, (Special Issue on Object-Oriented Experiences, Mohamed Fayad and W.T. Tsai Eds.), 38,10, October 1995.

[17] www.freshmeat.net

[18] O.- J. Dahl, E. W. Dijkstra, C. a. Hoare. Structured programming. 1978.

[19] http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html

[20] http://java.sun.com/docs/books/vmspec/

[21] http://java.sun.com/j2se/1.3/docs/guide/reflection/proxy.html