Annales UMCS Informatica AI 7 (2007) 107-115

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

Computer-based system for training and selecting mobile robot operators – evolving software tools

Krzysztof Sapiecha*, Mariusz Bedla, Barbara Łukawska, Paweł Paduch

Department of Computer Science, Kielce University of Technology,
Al. 1000-lecia Państwa Polskiego, 25 - 314 Kielce, , Poland

Abstract

A part of research on application of mobots (mobile robots) for watching real environment is the computer-based system for training and selecting candidates for mobot operators. The game played by the candidates is the main part of the system. The core of the game is touring simulator. Firstly, it was assumed that speed of simulation would be a crucial factor for the game. Hence, the initial version of the game was written in C++ for LAN. However, the experiments showed that promptness of the reaction of the game to the actions of the players was not so important. It turned out that wide and easy access to the game and its remote management (e.g. via Internet) are necessary. The current version of the game uses Java technology. Rewriting the game from C++ to Java created typical problems connected with moving an application from LAN to Internet. It was not a trivial problem. The paper evaluates and compares both versions of the game and describes problems with migration from C++ to Java.

1. Introduction

For years monitoring, watching and guarding have been human tasks. Recently, more electronic devices like motion detectors and cameras are used to support these activities. In many cases mobile robots (mobots) are also used for this purpose. However, human being (mobot operator) is still responsible for making decisions. Due to the fact, that guarded properties and equipment used are valuable, guards should be properly trained. To train and rank the operators a simulator and a game based on it were developed [1]. The simulator is similar to a flight simulator. During the game a mobot driven by an operator moves in virtual reality and takes photos. The task for the operator is to find all scene changes in limited time.

*Corresponding authors: *e-mail addresses*: k.sapiecha@tu.kielce.pl, m.bedla@tu.kielce.pl, b.lukawska@tu.kielce.pl, p.paduch@tu.kielce.pl

Requirements which the game should meet to be efficient enough were changing along with research proceedings. That brings about changes in the software. Firstly, it was assumed that speed of simulation would be a crucial factor for the game. Hence, the initial version of the game was written in C++ for LAN. However, experiments showed that promptness of reaction of the game to the actions of the players was not so important. It turned out that wide and easy access to the game and its remote management (e.g. via Internet) are necessary. The current version of the game uses Java technology. Rewriting the game from C++ to Java created typical problems connected with moving an application from LAN to Internet. It was not a trivial problem.

The paper evaluates and compares both versions of the game and describes problems with migration from C++ to Java. Section 2 shows how the research proceedings have modified functional requirements for the game. In sections 3, 4 and 5 architectures of the following versions of the game are described. Their advantages and disadvantages are pointed out. Section 5 includes comparison considerations for the architectures described. The paper ends with conclusions.

2. Game functional requirements

In the rapidly changing environment a mobot is driven by an operator. The goal of the operator is to plan a minimal mobot's tour to see every environment change before a deadline [2,3].

At the very beginning of the research the question arose whether a human being could achieve this aim or not? To answer the question the first research consisted in planning a tour of the mobot in a room (including positions where pictures should be taken) so that all changes in the room were detected within a time limit [1]. In the game a player or a group of players equipped with a map of the room used touring simulator to move across the room. Each player looked at pictures corresponding to what camera lens saw. He (they) decided which pictures had to be stored and what move would be taken next. His (or the group) task was to find every change in the room as soon as possible. Basic functional requirements for the game were as follows:

- fast simulation (real time),
- fast and flexible communication.

Because both: the mobot and the watched property may be expensive, the mobot should not be driven by an occasional person. A mobot operator should: detect all changes precisely (be perceptive), quickly (be responding fast but without emotions), reach a conclusion, be a good strategist and stress resistant. His qualifications should be as high as possible. It would be good to improve them, e.g. during a training course, to achieve the highest watching quality mobot operators can offer. Therefore, the next research consisted in checking to

what extent the game played in virtual reality can help in training and ranking candidates for mobot operators [4]. In this case, basic functional requirements for the game were as follows:

- fast and easy individual candidate data acquisition and management,
- variety of operating systems cooperating with each other (mobot, server, database).

As followed from experiments [5], that the game played in virtual reality could be used for training mobot operators. However, the training course is expensive, so we should try to find a cheap but still reliable method for selecting the best candidates from those available [6]. To expand research to a larger group of candidates, the game has to meet the following functional requirements:

- be accessible in the Internet,
- be endowed with tools simplifying administration of large amount of data.

Furthermore, it was not certain how realistic there should be the environment where the game was played. Hence, the following extra functional requirements were added:

- environment in which the game takes place should be more realistic,
- objects used during simulation should be realistic.

To meet these requirements virtual reality in which the game was played was enriched. The former geometric shapes were replaced by the objects like wardrobe, chair, etc.

3. C++/PVM based architecture

The first version of the game was PVM (Parallel Virtual Machine) [7] application written in C++ programming languages. Fig. 1 shows an overall architecture of the application.

Three kinds of processes: supervisor, game server and game client are distinguished. The supervisor process starts first. Then, it may start game servers as slave processes. The supervisor collects all data about movements and messages passed by players and puts them into logs so that one could analyse them later. The game server can connect or disconnect players. Messages related to player's movements are adequately interpreted and the results are sent back to clients. The game client is a program started using a separate computer by a player who wants to join the game. It was assumed that every player would download the whole scene where a mobot works. All pictures had to be rendered on the client side. The players could gather into groups and help each others.

The PVM library was created for sending messages. However, it is not fast enough, because of too many stages of sending information. A message is packed, then transmitted and unpacked. Theoretically, it is possible to create a

source code of an application using the PVM library which can be compiled on different operating systems, but in practice, it is almost always connected with preparation of modified versions of programs for each operating system. For our experiment we developed a version of the game for Linux. It also requires proper configuration of every node of LAN due to the fact that the game uses PVM. Additionally, maintenance of the game is difficult because updates are complicated and time-consuming.

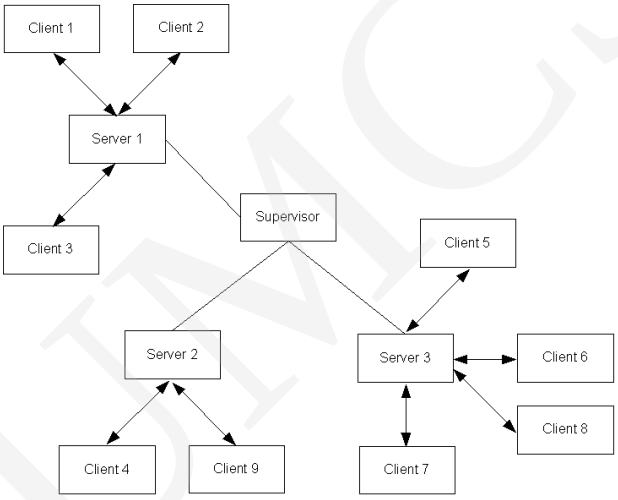

Fig. 1. C++/PVM version of the game

4. C++/QT based architecture

The second version of the game was written in the C++ programming language using the QT library [8]. It consists of the following applications: game server, clients and picture provider connected to a database or a mobot. Figure 2 shows connections between these applications.

The game server could be connected to real a mobot. At the very beginning of the research the Lego MindStorm mobot with a camera was used. Communication was based on sockets and TCP/IP protocol. An interface based on XML connected the mobot driver and the game server. The client command was forwarded by the XML interface to the mobot driver. The mobot approached the destination and took a picture.

Because the mobot proved to be inaccurate, slow and expensive we decided to create a touring simulator. Pictures taken by the mobot are stored in the database. The simulator takes a picture from the database on demand of a client who does not know which is a source of data. However, this method also proved be inaccurate and too time-consuming. Therefore, finally the picture generator is

used. It is a special program rendering pictures from the earlier designed virtual scene. The pictures are put in the database.

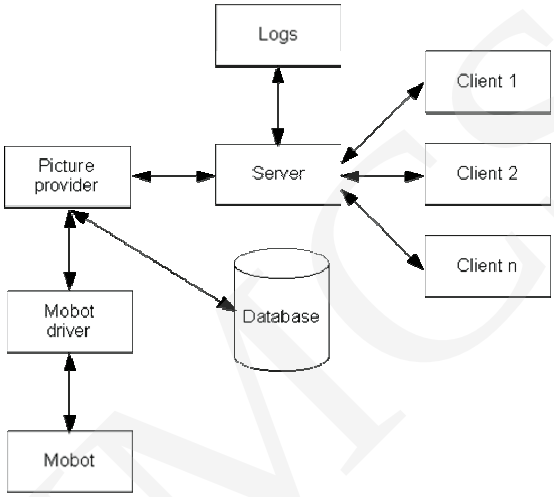

Fig. 2. C++/QT version of the game

In comparison with the earlier version of the game, here is the only game server. Players could not play in groups and are ranked on the basis of the number of their movements, photos taken, mistakes and change discoveries.

Using XML for communication between the elements of the game makes it possible to combine applications working using different operating systems and written in different programming languages. The QT library used in this project includes necessary support for sending the XML documents and simplifying communication by sockets. This version of the game works much faster than PVM one, but too many software modules make the game not as fast as it could be. Client applications still require individual upgrading on every PC.

5. Java based architecture

The current version of the game is implemented in Java [9]. Hence, it is widely available and more portable. The game consists of the following elements: game server, administrative tool and client in the form of an applet (Figure 3).

Due to the fact that all elements of the game are written in Java communication does not use XML but object serialization. Serialization makes it possible to send quite complex objects in an easy way. All data are stored in relational database which enforces data conversion while writing/reading between the database and the applications. In comparison with the previous

versions of the game this one is computer architecture and operating system independent. Updating the software can be done in just one place – on www server, where client applet is stored.

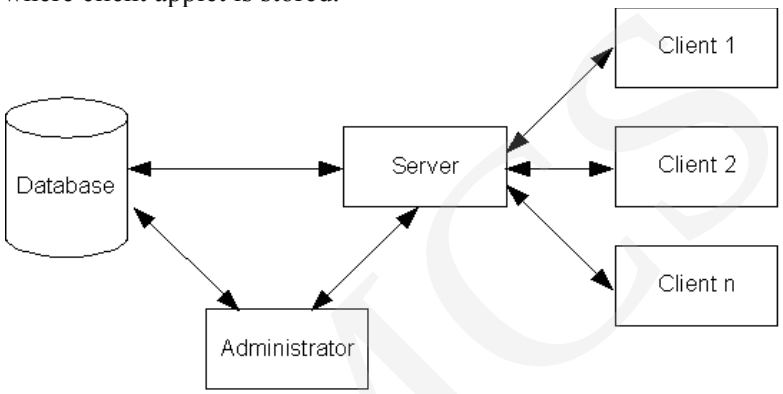

Fig. 3. Java version of the game

The server connects all players and manages the course of game. An administrative tool may be used for: scheduling of the game, monitoring a state of the game, managing sets of boards (adding, removing whole sets of boards and individual boards), managing players and writing results of an experiment into a file in XML, SQL or CSV format. XML format stores information about all events occurring when the game was played. It is used to transfer data between the touring simulator and the additional, external analyzing tools. In order to protect data against the system failure, or to move them to another system, an export to the SQL file could be done. Data in the CSV format are preprocessed. They contain statistics about a course of game.

6. Comparison considerations

The versions of the game differ in many points: architecture of the game evolved from LAN to Internet, from real time pictures acquisition to their database, from C++ to Java, from geometric to almost real objects and so on. Table 1 contains a brief comparison of these points.

To examine simultaneously more candidates the pictures taken by the real mobot were stored in the database. The simulator took a picture from the database on demand of a client. Finally the mobot was replaced by pictures generator. Introduction of pictures generator made it possible to modify any parameter of simulation, such as: environment, time, used objects, etc.

Table 1. Comparison of different versions of the game

Feature	C++ / PVM	C++ / QT	Java
Communication	PVM environment.	Sockets + XML.	Sockets + Serialization of objects.
Pictures storage	Only scenes and objects are stored. Original version of the game assumed that pictures would be created on the client side.	Initially, pictures were taken from the mobot camera. Next, from the pictures provider written in Java. Finally they were stored in the database.	Pictures are stored the in relational database.
Configuration	A bit difficult, this solution was prepared for local networks.	Much easier. It is sufficient enough if computers could communicate with each other using TCP/IP.	Easy. The player should just have a correctly configured web browser with the Java environment. The game administrator has additional options for game configurations.
Installation	Upload programs on each node belonging to the PVM pool.	Server runs on single PC. Client programs are installed apart. Sometimes recompilation of clients is needed for different versions of Linux system.	All applications require only the Java Runtime Environment. The player connects using the web browser.
Portability	Theoretically yes, but practically difficult, and we have never tried it.	Theoretically yes, but practically we have never tried it.	Quite independent of computer architecture and operating system.
Flexibility and universality	Whole system was quite hermetic.	Whole system was very flexible thanks to XML communication. It was possible to add different modules like mobot drivers or picture generators by one universal XML interface.	Java is computer architecture and operating system independent. Hence, all the applications are quite portable.
Time measuring	Centrally.	Centrally and on the client side.	Locally on the client side.
Code maintenance.	Complicated due to the fact that the distributed environment is used.	It is complicated because of multilevel logics and architectural distribution. The whole system was written in many programs languages.	All updating of client software can be made in just one place – on www server.
Speed	For the initial concept good, but not for large images.	Very fast, but too many software modules make the game not as fast as it could be.	Sufficient.

To expand the research using a larger group of candidates the game had to be accessible in the Internet, and modified to take into account features of the Internet. The influence of communication delays had to be minimized and course of game should be independent of discrepancy of time on the server and the client sides. A larger number of clients could cause larger server workload. To solve these problems a part of calculations was moved on the client side. Previous versions required knowledge concerning installation and initialization of the game. In the case of the Internet one could not assume anything about network and a user. The game written in Java requires only installed Java Runtime Environment and knowledge how to operate a typical web browser.

The game was rewritten from C++ to Java so that it was portable. In the C++ versions of the game one could use a mixed style of programming, object oriented and procedural ones. Java almost enforces the object oriented style: every variable must be an attribute of a class and every function must be a method of a class. In C++ programmer is responsible for creating and deleting objects. Not correct memory management may cause a leak of memory, which we experienced in the beta versions of applications. In Java so called "Garbage Collector" is responsible for cleaning up the memory. Therefore, the implementation of the game in Java was much easier. Due to the fact that Java is safer than C++ we made fewer errors and application could be developed in shorter time. Usage of Java made it possible to reorganize structure of the game. A client was not created as a standalone application but in a form of an applet located on the web page. This resulted in portability of the game, its independence of computer architecture and operating system, and made it possible to develop application which is easy to update and does not require sophisticated configuration. C++ language is not so flexible, but it has one great advantage – speed. However, for an application which use slow network connection it is not so important. Summarizing, comparison between Java and C++: Java programmer may think more about "space of a problem", than about "space of a solution", which cuts down time of development of an application.

To test how the operator behaves in different environments, two versions of picture generator were implemented. The first version makes it possible to create environment composed of only geometric figures while the other one generates more realistic pictures, supplemented with textures, shadows, etc.

7. Conclusions

At the very beginning one Lego MindStorm robot was used. However, using one mobot for experiment with a larger group of people was inconvenient. Due to this fact touring simulator was created. Initially it was assumed that the speed of simulation would be a crucial factor, so the first version of the game was

written in C++ with the PVM environment. As followed from experiments reaction time of the game on operator actions was not crucial. In the next C++/QT version of the game PVM communication was replaced by sockets, which simplified data transmission and management of the game. The third version of the game was written in Java. The game became accessible in the Internet and administrative tools were added.

Each solution described above is characterized by advantages and disadvantages. They were chosen on account of current research proceedings. The transformations of the game were not easy. It was only possible to use logics of application, algorithms and experience gained from the previous versions. It is simple illusion that moving from one kind of network to another or from one object-oriented programming language to another is convenient. Actually, such transformations lack any useful standards.

References

[1] Sapiecha K., Łukawska B., Paduch P., *Experimental Data Driven Robot for Pattern Classification*. Annales UMCS Informatica AI, 3 (2005).
[2] Dagnino T., *The Art Gallery Problem*. http://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/97/Thierry/thierry507webprj/artgallery.html
[3] Urrutia J., Watchman's problem. http://www.site.uottawa.ca/~jorge/openprob/Watchman/
[4] Sapiecha K., Łukawska B., Paduch P., *System wspomagający profilowanie operatorów mobilnego robota*. Zeszyty naukowe Politechniki Świętokrzyskiej, 43 (2005), in Polish
[5] Łukawska B., Paduch P., Sapiecha K., *An application of virtual reality for training and ranking operators of mobile robot*. Conference IBIZA 2006.
[6] Sapiecha K., Łukawska B., Bedla M., *Computer-based system for training and ranking mobot operators – selection procedure*. Conference IBIZA 2007.
[7] Parallel Virtual Machine http://www.csm.ornl.gov/pvm/pvm_home.html
[8] QT Library http://www.trolltech.com/products/qt
[9] Java Technology http://java.sun.com/