The architecture migration in multi-processor distributed systems

ROMAN GUZIK, JINDRICH CERNOHORSKY
Department of Measurement and Control
VŠB | Technical University Ostrava, Faculty of Electrical Engineering and Informatics
17. listopadu 15, 708 33 Ostrava Poruba
CZECH REPUBLIC

Abstract: - A possibility to design and to implement a software architecture which can be modified flexibly in case of requirements on extension, innovation or degradation of hardware architecture of distributed control system is discussed in this paper. With the aim the term migrating architecture is introduced which indicates the architecture enabling the transfer of software parts among single processors of a distributed system. The migration of software structures is realized only by the system configuration, not by the change of the source code of the application. The design patterns bringing the advantages of the object oriented programming into the network environment utilized today are described here from the point of view how to use them for implementation of the migrating architecture.

Key-Words: - Java, distributed control system, migrating architecture, design patterns, software, framework

1 Introduction
During the extension or innovation of a multi-level distributed control system forced by the changes in the structure of production processes for which the distributed system serves the question arises how great changes must be carried out in the control system and how much effort as well as costs the changes will require for reaching of the goals set.

For example let us consider the problem, when after extension of a production line the existing set of microprocessors at the lowest level of the control system will not be capable to ensure the computing performance necessary for control required. It would be nice to solve this problem by adding the additional micro-processors and by splitting up overall tasks among the processors of the extended set at the lowest level. An alternative solution would consist in transferring of some tasks from the lowest level to the processor on the higher level (mostly PC) and, herewith, to manage with the existing number of processors of lower level.

Another example could be the transmission of a visualization task from the working station onto PDA or mobile telephone in a limited form. The frequented requirement can be also the connection of diagnostic equipment to some processor without interruption of the system operation and transferring the computing performance of the equipment diagnosed to other processors within the network. The transfer of processing can be also utilized in case of some processor defect.

Then, in all mentioned examples, one of the first-row requirements laid on the software of the distributed system is, so that it could be easily modified and extended. Thus the question arises in which way to achieve this feature. The consideration concerning the way of solution of the above mentioned requirements led us to introduction of the idea „migrating architecture“.

Migrating architecture denotes an architecture enabling the transfer of software parts among separate processors without substantial effect on the source code of the application under keeping the sufficient computing performance and under saving the high degree of the system capability to be modified.

The software architecture enabling the migration of its single parts among some processors of the distributed systems on which such software is operated must be proposed in such a way so that the migration could be realized only by the system configuration, not by the changes of the application source code.

2 The distributed multi-processor systems
The distributed multi-processor systems utilize more processors interconnected mutually by the network, for the aim to achieve the common goal [1]. The simple applications can utilize one processor but for solution of more complex real applications it is necessary to use the multi-processor architecture. The architectures used in safety-critical control systems utilize, in addition, the various types of processors for ensuring the errors and defects arisen.
The primary problem in a multi-processor system is the achieving of coordination of separated subsystems. There are two primary communication management strategies – the centralized and the decentralized one. In centralized strategy one processor – master - is responsible for management of overall communication and the other processors only fulfill master’s requirements. This is also known as the master-slave architecture.

The basic advantage is the simple character of programs running on the slave processors. With the centralization of management we simplify the system capability to be modified and also the maintenance because all is concentrated in one place. However, in many industrial applications the large centralized control systems is not so effective as the distributed network of relatively simple computing entities.

In fully decentralized systems the computing load is split-up among all processors more or less equally. This architecture is called the peer-to-peer architecture. The coordination of separated subsystems is then more demanding, especially if a great information coupling exists between single processors. The modification of such systems represents then demanding problem what leads to undesirable quantity of software errors.

In industrial practice there exists the whole spectrum of applications from the master-slave up to the peer-to-peer architectures. The distributed control systems range from using very thin clients up to using very fat clients.

![Master – slave architecture](image)

**Fig. 1** Master – slave architecture

The decision whether to use the thin or fat client depends on many factors. The fat client is much more effective when in one node there is an excess of processor capacity and the system itself is relatively simple, very deterministic or, in its substance, only one-processor one. The thin client is much more suitable in applications when the achievable computing performance is distributed among several nodes and the system environment is so complicated that it is very troublesome to predict when the various events occur.

During the design of the multi-processor applications it is not quite easy to schedule unambiguously the separate tasks onto given set of processors. Despite the proper analysis particular problems will occur only during the time period of the implementation itself or in the time period of operation test.

For this reason it is necessary to design the software architecture as highly modifiable where the single processes can be transferred to the target processors without greater interference into the source code. If the application is proposed suitably, so the maintenance of the distributed system is more simple and so less costly. The advantage will occur especially in the subsequent modification or extension of the system controlled. The time necessary for carrying out of necessary adjustments and time for system modification will be decreased as well as the direct costs connected with it. Also the number of subsequent software errors caused by the maintenance will be reduced.

The presupposition for achieving of the given parameters is the suitable decomposition of the system on the entity of tasks without internal couplings and with the high cohesion within single tasks. For achieving the optimal decompositon it is necessary, on the basis of the data flow between individual tasks, to distribute these tasks to single processors as to reach the optimal cohesion and coupling criteria as stated above.

3 The design patterns for migrating architectures

The architectonic design can be achieved in more various ways. All possible ways solving the problem can be generalized into the design patterns organizing and ranking the sofisticated software structures for the purpose of achieving the system goals [2], [8]. The substance of a design pattern is the formalization of the approach to the solution of a general design problem. Among the most frequently used design patterns for the solution of tasks typical in distributed systems there belong Proxy Pattern, Broker Pattern and Dynamic BrokerPattern.

3.1 The proxy Pattern

The Proxy Pattern is the pattern applicable in the cases when the normal reference or pointer to the object is unsuitable or impossible. These are those cases when the object lays in other process space or even in other processor. This pattern separates the clients from their servers by creating their local representatives. The separation of the client and server means that the server lays in different address space than the client and, for this reason, it is convenient for implementation purposes useful namely in distributed systems. The various proxy objects can provide various access rights to data.
3.2. The broker pattern

The Broker Pattern makes the Proxy Pattern more accurate and represents the further step towards the separation of the client from the server. Broker is the object which knows the location of other objects in the system. Broker can have this knowledge in advance (in the time period of compilation) or can accumulate the information dynamically by means of object registration or utilize the combination of both procedures.

The primary advantage of the Broker Pattern is that it is possible to create the Proxy Pattern when the localization of the server is not known in the time period of the system compilation. That is why, the Broker Pattern creates the Proxy Pattern dynamically. Although many real-time systems use the commercial broker objects directly the made-to-measure implementations of broker objects are not infrequent.

The various types of migrating architecture

We have already stated that under the migrating architecture the architecture is understood which enables the transfer of specific software parts among single processors without substantial effect on the source code. From the viewpoint of the design it is possible to divide the migrating architecture into three basic sub-groups.

- **The static migrating architecture** – the application is distributed on single processors on the basis of the optimization criterion evaluated during the system compilation. The change of configuration is possible only after repeated compilation. It causes nor the deceleration of the application nor requires more memory during runtime.
- **The dynamic migrating architecture** – the single components of application can be distributed to single processors during the runtime. This could be important especially in case of continuously running systems. The transfer of components on the target processors and initialization of components on target processors cause the time delay. So this way can be utilized in case when the time delays have no affect on the application performance.
- **The pseudo-dynamic migrating architecture** – the single components of the application are loaded on single processors during the system compilation. However, the processes are redundant, i.e. the same processes can be started up on more processors. Then, according to the system load, the application itself will decide on which processor the given process will be started up. In this case the greater requirements are laid on memory, however the overhead costs connected with the transfer of components under run falls out.
4 Java – the language for realization of the migrating architecture

The features of programming language for implementation of migrating architecture are very important. With respect to the requirements mentioned in the previous text it is possible to formulate the following requirements on the programming language to be suitable:

- Language independence on the operating system
- Possibility to use the same programming language at all levels of distributed systems
- Support of the network programming
- Support of the real-time programming

These requirements, to a great extent, are met by the programming language Java.

4.1 Language independence on operating system

For the reason of great variability of operating systems used in the distributed control systems it is necessary to lay the requirement of independence of the programming language on the operating system used. To a certain extent, the language Java meets this requirement. The program written in the Java language can be started up on each processor on which the Java virtual machine (JVM) runs. At starting up, the runtime system of JVM

- reads-in all necessary classes by class loader
- starts up the byte code verification unit which tests whether the dynamically loaded classes do not disturb the language limitations of Java.

In the time being, JVM is implemented for all operation systems currently used.

4.2 The utilization of the same programming language on all levels of the control system

With respect to the necessity to transfer the components of migrating architecture among single levels of the distributed control systems the existence of the same specification of programming language for all processors of the distributed system is necessary. JVM is implemented in a great number of embedded equipment units, servers, mobile telephones and PDA so there is for disposal great selection of the equipment units which we can use in various applications.

4.3 The support for network programming

The support for network programming is one of the key aspects for building distributed control systems. The Java language supports the network communication with help of datagrams and sockets. It supports also the RPC technology (remote procedure call) by its purely own Java technology RMI. However, Java is not limited only by mentioned technologies. The communication with applications written in other programming languages using CORBA is also possible. This characteristics is the key one for safety-critical applications where, for preventing from errors, it is suitable, besides various types of processors, to use also various programming languages.

4.4 The support for real-time programming

Most of distributed control systems are, in their substance, the real-time systems. Therefore it is suitable so that the programming language used for implementation on these systems would have the real-time characteristics implemented directly in the language definition.

The existing specification of JVM does not meet fully the real-time requirements. However, the remedy of the specification of the Java language [3] runs for the real-time utilization. The work group Real-Time Java Expert Group (RTJEG) is working on extension of the specification of the Java language and specification of the Java virtual machine. In addition, it defines also the application programming interface (API) which would enable to create, verify, analyze, perform and control the Java threads with the help of exact conditions including the time limitations. Such threads are as the real-time
threads. The RTJEG specification enables also the possibility of selection of the scheduling algorithm according to the particular needs of the application.

Also the commercial products for utilization of Java in the RT applications arise. The example is the product of the Swiss firm Esmertec [4] called JBed oriented on the embedded real-time systems. The other product utilizing Java for the embedded system is the product of the firm aJile systems [5] which develops the processors comprising the Java technology and a real-time kernel. Compared with JBed where the operation system is implemented only as software and utilizes the available processors, the firm aJile systems selects the way of the Java support directly on the hardware level.

5 The suitability of the Java technologies for realization of migrating architecture
Next we shall discuss those technologies of the Java language which seem to be key ones for realization of the migrating architecture. This is especially the question of support of network programming [6], [7].

5.1 The communication with help of datagrams
The communication with help of datagrams is used for sending the non-critical data. The sending of datagrams is quick but it is not guaranteed here that the separated datagrams will be delivered in the same order in which they were sent and also that all of them will be delivered. The datagrams are sent with help of the class DatagramPacket in the constructor of which we enter the parameters of the bytes field. From the viewpoint of the migrating architecture the utilization of datagrams for visualization of non-critical data is possible.

5.2 The communication with help of sockets
For reliable network connection the classes Socket and ServerSocket are used. The class Socket is appointed for realization of the client. It enables the connection to an Internet host on a given port and subsequently reading and recording data by means of classes InputStream and OutputStream of the package java.io. The class ServerSocket enables to implement the server which receives the connection from the clients. The port number is passed to the constructor on which the server is to be listened to. The waiting for a client is ensured by the method accept() which returns the instance of the class Socket to which the client is connected. Both classes Socket as well as ServerSocket utilize the class InetAddress which represents the Internet address.

The migrating architecture can utilize the sockets for communication with the equipment units on which, for the reason of limited memory or processor capacity, the more advanced technology is not implemented.

5.3 The serialization
The serialization extends the kernel of the Java Input/Output classes by the possibility of depositing the objects into the stream bytes and also their full renewal from the streams. These objects can be deposited and subsequently utilized also by other application. They can be also sent on network with help of the sockets. The serialization of the objects is used also in case of RMI. In case of the object which we can deposit into the streams and utilize again subsequently we have to implement the interface Serializable or Externalizable.

The interface Serializable is used in case when we leave the depositing of the objects attributes as well as the whole hierarchy of subordinated objects directly on the optimized JVM algorithm. In case when we do not want to serialize all members data of the object, for example for the security reason it is possible to use the interface Externalizable. In this case the serialization must be done by the programmer himself. The serialization of the object can be customized using the interface Serializable and by keyword transient applied on the object if we request the own method of serialization.

The serialization is one of the most important characteristics for implemention of migrating architecture. Thanks to serialization it is possible to transfer the processes among single processors in the network.

5.4 The remote method invocation
In creation of distributed applications based on the client-server technology the remote objects provide the client with the possibility to request from the server the direct reference to its object. In traditional data communication model the client gets from server only the data. In RMI communication the client can handle the data of the server with help of methods of remote object in the same way as if they were the methods of his local object. This solution is more flexible, the client and server become more independent. The server can change the way of data representation without necessity of the modification of the client. Only changing the server methods under saving the object interface on which the client has the remote references is sufficient.

With help of remote objects the migrating architecture can select the best processor on which the given task will be started up.

5.5 The general solution of the remote objects
In this paragraph we’ll shortly define several terms important in respect to the implementation of the remote objects which are crucial as to the migrating architecture. In a general architecture of remote objects the communication between the object representative and its
Implementation is mediated by *Object Request Broker (ORB)*. The specification of the remote object interface can be expressed by means of the *Interface Definition Language (IDL)*.

### 5.5.1 The RMI technology

**Definition of the terms**

- **Remote object** - object the methods of which can be invoked (called) from further JVM
- **Remote interface** – declaration of methods of the remote object with the help of the Java interface
- **Remote method invocation RMI** - invocation of the remote object method through remote interface. This invocation has the same syntax as the invocation of the local object method.

### 5.5.2 The RMI architecture

The RMI architecture implemented in JVM utilizes the standard mechanism for communication with the remote objects:

- **stub** (representative of the remote object on the client side)
- **skeleton** (skeleton of the remote object on the server side)

The relevant representatives of the remote object (stub) and skeleton of the remote object (skeleton) are generated by the *rmic* compiler.

### 5.5.3 The garbage collection of the remote objects

In the distributed system as well as in the local system the JVM cancels automatically the remote objects to which already no client is referring. This feature of Java is the most important compared with other technologies realizing the remote objects invocation.

### 6 Conclusion

It is showed that the programming language Java can be used for development of the so called migrating architecture by which we indicate the architecture enabling the transfer of software parts among single processors of distributed control system without substantial effect on the application source code. The necessity to solve such a problem seems to be the basic one in distributed multi-level control systems when, in exclusive cases, it is necessary to transfer timely or durably the solution of one or more tasks from one node of computing system to other processors. This transfer can be carried out both on the same control level, if need be also between various control levels. In the time being, the programming language Java is equipped not only by suitable technologies for solution of such tasks but there exist also relevant developing environments in sufficient number for implementation on various hardware platforms as well as environments of various operation systems.

**Acknowledgement**

The work and the contribution were supported by the GACR project “Pilot extensible assistance framework for control and information systems development ” ( No 102/01/0803 ) provided by Czech Grant Agency, which is gratefully appreciated.

**References:**

2. [Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Reading, MA: Addison-Wasley, 1995.]
3. [Real Time Specification for Java, USA, Sun Microsystems, Inc., 2002]
5. [aJile Documentation, USA, aJile Systems, Inc., 2002]