Modelling the Replication Management in Information Systems

Cezar TOADER [1], Rita TOADER [2]
[1, 2] Technical University of Cluj-Napoca, Department of Economics,
North University Center in Baia Mare, Romania
cezar.toader@cunbm.utcluj.ro, rita.toader@cunbm.utcluj.ro

In the modern economy, the benefits of Web services are significant because they facilitates the activities automation in the framework of Internet distributed businesses as well as the cooperation between organizations through interconnection process running in the computer systems. This paper presents the development stages of a model for a reliable information system. This paper describes the communication between the processes within the distributed system, based on the message exchange, and also presents the problem of distributed agreement among processes. A list of objectives for the fault-tolerant systems is defined and a framework model for distributed systems is proposed. This framework makes distinction between management operations and execution operations. The proposed model promotes the use of a central process especially designed for the coordination and control of other application processes. The execution phases and the protocols for the management and the execution components are presented. This model of a reliable system could be a foundation for an entire class of distributed systems models based on the management of replication process.
Keywords: *Information Systems, Services, Reliability, Fault-Tolerance, Replication Management*

1 Introduction

This paper presents the development stages of a model of a distributed system in which the primordial abstract elements are the process and the connection (link) between processes. This paper deals with the problem of communication between the processes within the distributed system, with the concept of message exchange between processes, and also presents the problem of distributed agreement.

This paper presents the importance of two concepts: the logical time of a distributed system and the causal order of messages. All of these concepts are necessary to define a model of a distributed system in which some processes have the role of management of the operations carried out by other processes which only have an execution role.

After presenting of the processes of the system components, there is an abstraction of the exchange of messages between the processes of the distributed system. In the next step, an abstract model of a distributed system is proposed and analyzed, in which there are modelled several worker-processes controlled by a manager-process, which runs on different host than the worker's. The model described here forms the basis for the development of a distributed system tolerant of faults, which uses the replication of operations.

Based on the abstract model of the distributed system previously presented, a replication model is proposed, for operations and data based on the structure of a distributed system containing several worker-processes. After the presentation of a list of objectives for the fault-tolerant system, a logical separation of management operations from the execution operations and data storage is proposed.

Web services are increasingly used in distributed systems, applications and services of outmost importance. For distributed systems, the architecture oriented on services, SOA - Service Oriented Architecture is already accepted as being the architecture able to interconnect applications running on different operating systems and facilitates the complex interactions between autonomous systems and heterogeneous ones, either within organizations or between organizations in relations of B2B (business-to-business) type. Web services allow software applications found in different organizations (company) to

interact with each other, even if those organizations use different hardware systems, or operating systems and even different programming languages. Web services are able to standardize and improve the effectiveness of business activities on the Internet by automatically invoking operations which otherwise should be invoked manually by a human operator. So, Web services allow direct interactions between computers located in different organizations.

Based on Web services it is possible to build technologies to support interoperability with Decision Support Systems. Important aspects for healthcare systems are described in [1], where the proposed technology is meant to raise the interoperability degree between different medical information systems is described. Web Services ensure communication between medical units automatically with minimum human intervention, which is an essential requirement when designing applications for users in the medical domain.

The design of applications based on Web services can take into account important aspects in creation of new levels of abstraction, giving a special importance to scalability and adaptability. Due to a growing importance given to resource-oriented Web services, the authors must implement the principle of organizing the algorithmic resources in multi-level hierarchy and the principle of uniform access to resources. In order to implement these principles, an architectural model of Web applications is promoted in [2].

2 Availability and Reliability of Distributed Systems

The failures and damages of web applications may lead to incorrect processing or even to the system failure in the business of e-commerce type, e-banking or other systems based on transactions. One of the important causes of the services' interruptions is the so-called „server falling". Therefore, there are becoming increasingly important the techniques which provide fault tolerance and the provision of the services on Internet even

in the conditions of damages to the servers.

For many distributed systems based on Web services there are a large number of client applications already installed and is very complicated the modification of all of these applications. This is the reason why the service providers search for specific fault tolerant solutions, called client-transparent, which do not require any special action on the part of the client applications or any amendments of them. This transparency on the part of client application is an important requirement for both the application itself, as well as for the operating system in question.

In complex applications, Web services should be connected to other services with a view to form composite Web Services and complex architectures based on services (SOA, Service Oriented Architectures). If a component in the chain of services is not available or is not reliable, then the entire system is affected.

A correct service is the one which implements the function of the system concerned. When the service provided by the customer differs from the correct service, it means that there is a failure of the system. This deviation means that the service does not comply with its well-defined specifications. An error is that part of the system status which may cause a failure. The cause of the error is a fault. A fault can be active or not. When it is active, the fault produces an error in the system. Serious errors lead to the system crash [3].

Web Services can raise new problems in the computer systems of the organizations:

- Faults existing in the information system of an organization may adversely affect the information system of a partner organization;
- Data consistence, integrity and confidentiality are more difficult to keep;
- Lack of availability, reliability and data security may cause damage to the relations between a company and their customers, suppliers and partners.

These problems are becoming more important and more demanding as the activities in the information system necessary for the conduct of the business are becoming more automated, as the Web Services calls for other Web

services and the business activities require more steps.

The system availability refers to the ability of a system "to be ready to provide a service correctly". The system reliability refers to the ability of a system "to be able to continuously provide a correct service". Dependability is a more comprehensive concept which incorporates several components: availability, trust, security, confidentiality, integrity, maintainability [3].

A very important means to achieve the reliability is the fault tolerance. This refers to the techniques meant to give the system the ability "to provide a service correctly even in the presence of errors".

In the systems considered reliable, data replication is a technique that has been widely accepted that allows the avoidance of system crashes. Thus, the architects of the system implement a service within a distributed system using a group of servers, independent from a physical point of view in such a way that if a part of them cease to operate, the remaining servers have the ability to provide the service in question to the clients.

Replication protects an application running on a server against the faults, so that if a replica becomes inoperative, other replica is available to provide that service to customers. The most frequently used replication strategies are classified as follows: liabilities, active and semi-active. An overview of these strategies can be found in [4].

2.1 Traditional Replication Techniques and their Limitations

Data replication consists of maintaining several copies of data, called replicas, on separated computers. Replication is an important technology in the field of distributed services. Replication improves the availability of the data by means of enabling the access of users to duplicate data in the close vicinity, even when some copies of the data requested are not accessible.

Replication improves the system performance by reducing the waiting times when to a user, it is offered data located in the duplicates from immediate vicinity, thus avoiding the large

distance access. Furthermore, there is an increase in performance due to data replication, which is manifested by simultaneous serving of multiple client applications.

Traditional techniques of replication are aimed at maintaining the consistence between the main package of data and a single copy, and the client applications could "see" a single set of data with high degree of availability. The basic concept is as follows: the access to replicated data is locked until they are up-to-date, and this update is proven within the application. This is the reason why such techniques are called "pessimistic".

The algorithms use at least two replicas of data. In many cases, it is necessary to choose one of these two replicas which receives the role of the main replica. After changing any data, the main replica transmits the main changes to the other replicas and only after this data synchronization, the client application is allowed to carry out new operations on data.

In the case in which the main replica becomes unavailable, between the remaining replicas, it should be chosen one to become the main replica. These techniques of pessimistic replication can be used successfully in the local networks, where the transfer speeds are relatively high, latency is relatively low and system crashes are relatively rare.

The replicated entity could be of several types: data object, file, data structure, and service. Therefore, the replication techniques may be different in order to be able to operate with specific components. There are at least two categories of replication techniques:

- Techniques targeting the replication of databases;
- Techniques targeting the replication of objects and processes in a distributed system.

The techniques of the two categories above have many similarities, but also important differences [5]. Due to the continuous progress of Internet technologies, it appears the tendency to apply the algorithms specific to pessimistic replication to wide area networks (WAN).

However, in this case, there are not expected

the same performance and the same data availability as in the case of local area networks.

In the first place, the Internet is relatively slow and does not provide the same reliability and the same data availability as local networks [6]. These problems are more evident once with the more and more usage of mobile computers with intermittent connectivity.

If an algorithm to woof pessimistic replication attempts to sync with a site which became unavailable (inaccessible), it will be in a locked status (locked pending an external event). Furthermore, there is even the possibility of data corruption because, in the event of a network break, the recovery time is not predictable and thus it is impossible the right choice of a main replica [7].

Secondly, the algorithms of pessimistic replication are not easily to scale for being used in WANs. It is difficult to build on a WAN a pessimistic replication system, having a large size, in which frequent updates of data appear because the increasing number of served sites leads to the degradation of important performance parameters: response times and availability of data [8]. That is why many Internet services adopted the "optimistic" replication techniques.

Thirdly, certain activities of the human users require data-sharing. In many engineering areas and especially in the software engineering, the specialists often work on punctual problems, well delimited and specified, in a particular isolation until the completion of the work in question. Therefore, it is better to allow independent updating of data in centralized data deposits of the organizations and the subsequent solving of conflicts appeared, instead of blocking the access to a specific set of data until a certain human operator finishes editing the data [9].

2.2 Optimistic Replication and its Features
Optimistic replication signifies a set of techniques for the efficient sharing of data in mobile or large size working environments. The essential feature that distinguishes the algorithms for the optimistic replication from those of pessimistic replication is the manner

in which the control of concurrent access is approached.

The pessimistic algorithms coordinate the synchronization between replicates during access and block the new requests of users for the duration of an update in progress. The optimistic algorithms allow access to data even if it has still not been completed the synchronization of replicas on the basis of the optimistic hypothesis that conflicts will appear only rarely, or not at all. The updates are sent to replicas in the background, and the occasional conflicts are solved when they occur. This optimistic way of seeing things is not new, but was widely spread as the Internet and mobile technologies have become more and more used.

As compared to the traditional techniques of pessimistic replication, optimistic replication promises a high degree of system performance and of data availability, but accepts for a limited duration some differences between the main data and their duplicates, this inconsistency being solved in due time.

The optimistic algorithms have several advantages compared to pessimistic ones in the field of distributed applications over the Internet. In the first place, applications are progressing even if the network connections or the sites are unreliable.

Secondly, the optimistic algorithms are flexible as regards the network in the sense that the used techniques propagate operations reliably to all replicas, even if the graph of the network is unknown and variable.

Thirdly, these optimistic algorithms are easily scaled in the case of a large number of replicas, because they require a smaller synchronization effort between sites.

In the fourth place, the optimistic algorithms allow sites and users to remain autonomous, meaning that it is possible to add a replica to the existing ones without the need of changes to the existing sites. In the fifth place, the optimistic algorithms enable the asynchronous collaboration between users, as in the case of CVS systems [9].

In the end, there has to be said that the optimistic algorithms give a fast response to client applications as they carry out the

updates of data as soon as they have been requested.

These advantages require a cost. Any distributed system is faced with the problem of a compromise between the consistent data and their availability [10]. In the early stages of work where the pessimistic algorithms are waiting to achieve a certain status of the system, optimistic algorithms allow the continuation of the operations with the client applications, but in the background, they are carrying out other operations.

The optimistic algorithms have to cope with the situations in which there are replicas different from each other, which may lead to conflicts between concurrent operations. Therefore, they are suitable only to applications that can tolerate occasional conflicts and data inconsistency for a limited time.

2.3 Elements of Optimistic Replication

Any system of replication operates with the concept of minimum unit of replication. These units of replication are, practically, objects. A replica is a copy of an object stored on a site, in a computer. Replicas of several items are stored on a site. Every object is independently managed in accordance with the mathematical algorithms of replication used.

Some algorithms distinguish between the sites that have permission to modify the objects contained, called master sites and sites that only store replicas accessible only for reading. Usually it is noted with R- the total number of replicas and with M- the number of master replicas of the same object. A case is the one in which M=1 and R = M, meaning that there is a single master site that contains a master replica for every object.

Other important components of replication refer to the operations and the manner of spreading operations on replicated systems. A system of optimistic replication must allow access to replicated objects even if one of the replicated systems is inoperative or disconnected.

An update of an object in the system bears the name of the operation. In the case of optimistic replication, the operations differ from traditional updates in data bases, because the operations are propagated and applied in the background, often after an answer was submitted to the client application which has initiated the request.

An operation can be regarded as a precondition for the detection of conflicts combined with a command for updating an object. The nature of the operations differ from one system to another. Many systems only supports the update of the whole object in question. These are called systems based on the status transfer (state-transfer systems). Other system allow the detailing of the update operations and are called systems based on the transfer of operations (operation-transfer systems) [11].

To update an object, the user requests a specific operation on a specific site. It performs the operation at the local level and then allows the user to continue to work relying on the fact that that task has been completed. The site calls in the background for other servers (possibly data bases) and requests the remote operations, to propagate the request received. These systems give eventual consistency as the status of replicas will converge, possibly, towards consistency.

This guarantee of consistency is practically very low. It may be considered sufficient for many applications of optimistic replication but, however, some systems offer better guarantees like, for example those in which the status of a replica is not updated more than a certain period of time. An operation is registered in order to be propagated later on other replica servers.

Due to the propagation in the background, the operations are not always received in the same order on all sites. Each site must reorder the received operations in order to produce results equivalent to those offered by other replicas from the system.

A "replica" of the system must reorder repeatedly its operations until it reaches the concordance with the other replicas from the system on the final order of operations. The term used for the policy of ordering of operations is "scheduling".

Without a coordination of the sites,

established beforehand, it is possible that multiple users to update the same object at about the same time. A relatively simple solution may be the arbitrary choice of an amendment requested by a user and ignoring the others. However, such a policy for the treatment of competing requests leads to the appearance of lost updates. These lost updates are not desirable in many distributed applications.

A better solution for the treatment of such problems consists in detecting the operations which are in conflict and solving them by renegotiating their succession. A conflict occurs when the conditions necessary for an operations are broken, even if the operations were carried out according to the system scheduling policy. A fault tolerant architecture based on Web services is presented in [12], where system functionalities, such as replication, fault management and client transparency, are analyzed.

3 Modelling the Replication in Distributed Systems

3.1 Process, Messages, Manager, Worker
The system abstraction should begin with the abstraction of the physical infrastructure which will be used by the system. The definition of the system model implies, in the first place, the description of the relevant elements with their specific properties and specifying the manner in which these elements interact with each other.

In this paper, there are used two abstract elements which allow the representation of the physical infrastructure of the system: the process, and the connection between processes.

Within the framework of a distributed program, the process is abstracting an active entity which carries on its activity after a certain algorithm and performs a certain processing of specific data. The process may represent a computer, a processor of a computer or a thread.

The processes should be able to cooperate for the fulfilment of common tasks. Therefore,

they should exchange messages between them. The messages between processes are possible only if there is a specific physical link between processes.

The connection is abstracting at the physical level (and also at the logical level) the link between the processes and lies at the foundation of the communication between the processes. Organized in a certain way, the links form network. The communication between processes involves the following components: the message m, the sender process S, and the receiver process R, as seen in Figure 1.
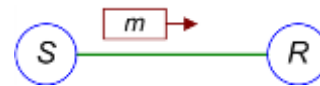

Fig. 1. The communication between processes

The description of a distributed system requires a multitude of properties of these processes and connections, as well as the manner in which these items operate (or cease to operate) under certain conditions in the working environment (Figure 2).
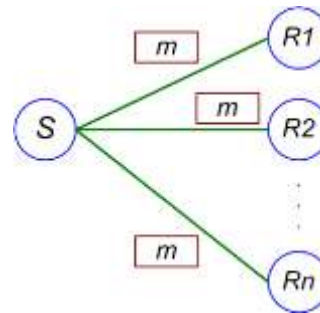

Fig. 2. One sender process, S, and several receiving processes, R1…Rn

In many cases it is important to locate the processes, i.e. the elements that host the processes and ensure their conditions of progress. For this we use an abstract element called "host".

A usual case is that in which the host is the computer on which a certain process is carried out (along with other processes in the operating system).

The importance of a host is given by the fact that it provides certain environmental working conditions for the hosted process, and the

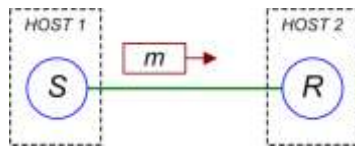process is influenced by the conditions offered by the host, as seen in Figure 3.

Fig. 3. Processes located on different hosts (S = sender, R = receiver)

Using the abstract elements mentioned until now (process, connection, the host) may be shape distributed systems most complex in which a process communicate via messages with several processes, some of them located on the same host, and other located on different hosts, as seen in Figure 4.
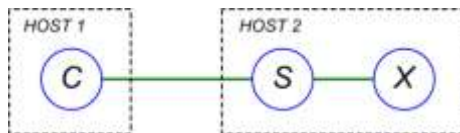

Fig. 4. The process S communicates with more processes (C = client, S = server, X = execution process)

There can be build systems in which each process has a certain role in the framework of the distributed system, and in order to fulfil the role, the process must be able to carry on specific activities. In these cases, it is important the nature of the messages received and sent between processes.

For instance, there may be built distributed systems in which a process C, with the role of client, sends a message to a process S, with the role of the server, by asking him to carry out certain operations and to give him a reply message.

The process S has an internal processing part, but it may send a message to another process X, with the execution role, requesting it specific operations. In this case, the process must communicate with both the process C as well as with the process X (Figure 4).

In the case of some complex distributed systems, more processes can be defined, with the role of execution, noted, for example, X1, X2 and so on, located on different hosts, which carry out specific activities after receiving specific message from a process

with the role of manager, noted, for example, with M, as seen in Figure 5.
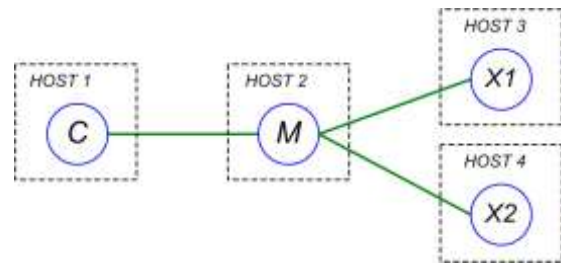

Fig. 5. The process M plays the role of manager for the worker-processes X1, X2

3.2 The Matter of Distributed Agreement
A model of distributed system is abstracting the interactions within the system. These refer to the cooperation between processes. This cooperation may be shaped as a matter of *distributed agreement*. Between the processes of a distributed system there must be an agreement on, for example, the interpretation way of a certain set of input data, the mode of expression of identity of the processes, performing a particular sequence of operations from several possible variants, respectively, achieving a consensus, etc.

In a distributed system, it is possible that there is an agreement between the participating processes relating to a specific task, which must take place only if there are met several conditions expressed by different data existing in the participating processes. In the case in which there are not fulfilled all the conditions laid down, then the participating processes "come to an agreement" that the operation in question does not take place. This form of agreement between the processes of a system is used in the case of transactions.

In a distributed system, the participating processes in a distributed program must be agreed upon the operations that must be carried out, but there must be also an agreement on the order in which the operations must be carried out, named the total order broadcast. This form of distributed agreement is one of the basic techniques of systems, which use data replication processes in order to realize the fault tolerance.

3.3 Processes and Operations

For a process, let's consider the abstract element called operation. Within a distributed system, a participant process transmits and receives messages from other participating processes. The nature of the messages may cause the execution of operations in different ways in the process.

An operation consists of the following steps:

- Receiving a message sent by another process;
- Execution of calculations at the local level;
- Sending a message to another process within the system.

Receiving and transmitting messages are global events in the system because there are more participating processes, at least two. The execution of the calculations at the local level does not imply a direct participation of other processes, therefore there are internal events of the process in question.

The process can have several modules, each of them can participate in a stage of an operation: to receive a message, to execute some calculations, to send a message to another process. These modules have specific roles and cooperate each other.
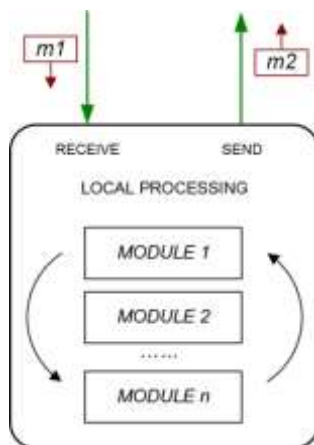

Fig. 6. The operation steps in a system process

The process may have an organization on several levels, a module being placed, from the point of view of the logic of programming, on a certain level. A complex process is that in which there must be run a multitude of operations and, therefore, at the software

level, the process has a multi-level structure called "software component stack".

3.4 Levels, Components and Events

At each process, the stack of software components contains one component for each level from the logical structure of the process. The top level is called the Application level, while the lower is the Network level. In the model of distributed system, the elements which are abstracting the distributed programming are in the middle of the stack levels.

The software components located on different layers in the same stack communicate with each other via the events, as seen in Figure 7. Each component is, at a given moment, in a given status. The receipt of an event triggers a transition to a different status of the component.

The event is a grouping of information placed in a well-established structure, previously defined and known by the components that receive and/or transmit the event. The rules for the communication between the components determine certain types of events. Thus, each event has the following characteristics:

- It is of a particular type;
- It has a certain source (the component which sent the event);
- It has a series of attributes;
- It has a destination (it is carrying certain information for other components).

The notation for an event used in this paper, starting from here, is:

$$\{ \qquad\qquad\qquad comp, EvType \mid attr1, attr2 \ldots\} \qquad (1)$$

In the equation above the used symbols are:

- *comp* is the source component of the event;
- *EvType* is the type of the event;
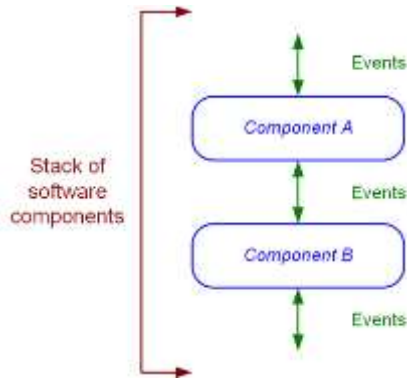- *attr1, attr2* … etc. are the attributes of the event.

Fig. 7. The model of a process based on components and events

Observations:
- Several components can use events of the same type;
- Several components can use the same event;
- An event received by a component can be retransmitted without changes to the other components, keeping or not the information about the original source.

An event received by the destination component is processed by a software entity especially designed for this operation, called handler. In the pseudo-code that describes what operations must be carried out on the basis of the receipt of the event, there will be used an instruction especially designed for this, that indicates the event and the instructions which must be carried out due to the occurrence of the event in question: upon event.

The processing of an event has a number of features:
- An event triggered by a component is processed only if the process which all the components are part of, are carried out correctly.
- The processing of an event can trigger the creation of new events by the same code or by different components.
- The destination component of an event can filter explicitly events. This will be indicated in the pseudo-code by clause such that.
- The events of the same components are processed in which they have been triggered.

- The events exchanged between the components in the same stacks of components are listed in FIFO mode (first-in-first-out).

The interfaces of the components contain two types of events: requests and indications. The events of type request are used by a component in any of the following purposes:
- To invoke a service provided by a component;
- To signal a condition of other components.

From the point of view of the treatment on the component software, the events of request type are input events (input).

The events of indication type are used by a component in any of the following purposes:
- To deliver a signal;
- To signal a condition of other components.

From the point of view of the treatment on the software component, events of indication type are output events (output).

Consider now a process which contains several software components, each of which is located on a certain layer in the logical structure of the process.

Each level communicates with the top level (if it exists) through a series of events of request and indication type. On the other hand, each layer communicates with the lower level (if it exists) through another series of events of request and indication type.

The propagation of the events between the levels of the process, applications from top to bottom and, respectively, the indications from bottom to top is shown in the Figure 8.
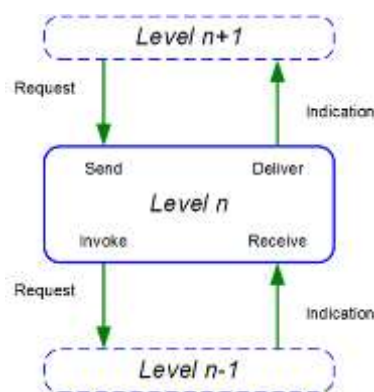

Fig. 8. The communications between the process levels

At a given level N, the execution consists of the following steps:

- The Message Request released by the software component on the top level, the level n+1, is received at the level n where it triggers the Send procedure for sending a message further toward the lower levels.
- The component on the n level executes the Invoke procedure in order to invoke the services from the lower hierarchical level, n-1, using for this Request events specific to lower level.
- The messages of type Indication sent by the lower level are received at the level n in the Receive procedure. They must be passed on to the top level.
- The message of type Indication, if it fulfils the conditions necessary for the safety of the process, is delivered to the upper level in the Deliver procedure.

The messages of type Request or Indication, which are transmitted between the levels, do not always have a data payload. Sometimes, they indicate only the conditions for the synchronization of the levels. For example, the top level can distribute a specialized message to the other levels to indicate that a particular processing phase has been completed and that it will move to the next phase.

3.5 The Communication between Processes
Consider now the two processes S (sender) and R (receiver) between which there is established a connection. Over this connection, the processes send their messages.

To build the model the communication between processes, consider now the point-to-point connection between two processes as an object called *conn*, which is an instance of a class called *PointToPointConnection*.

At the level of the object conn there are taking place two distinct events, one at the end of the process from the source process S and the other one, at the end of the destination process R.

The event which models the sending of the message m by the source process S is:

$$\{ \qquad\qquad\qquad\qquad conn,$$

Send | R, m } $\qquad\qquad\qquad$ (2)

The event which models the receipt of message m to the process R is:

$$\{ \qquad\qquad\qquad\qquad conn,$$

Receive | R, m } $\qquad\qquad\qquad$ (3)

In (2), the attributes of the event are: the recipient R and the message m. In (3), the attributes of the event are: the transmitter S and the message m.

In the model of distributed system analyzed in this paper, the correctness of the communication between processes assumes the compliance with the following list of properties.

The properties of secure communication between the correct processes are below:

- *P1. Reliable delivery of messages*. If the process S, which shall be carried out correctly, sends a message m to the process R, which also shall be carried out correctly, then the process R possibly receives the message m.
- *P2. The messages are not duplicated to the source*. A fair process, at any stage of its execution, sends a message only once.
- *P3. The messages are not created at the recipient*. If a message m has reached the destination process R from the S process, then the message m has surely been sent before the process R, by the process S.
- *P4. The messages are received in the strict order of their arrival* (FIFO). If the process R has received from another process S firstly the message m1 and subsequently the message m2, then the correct operation of the R process means taking in order of messages: firstly m1 and then m2.
- *P5. Compliance with an agreement at the level of the entire distributed system*. If a message m was delivered successfully to a process R that is correct, then the message m is eventually delivered successfully to any correct process in the system.

The correctness of the processes and communication is the basis of the proper functioning of the distributed system.

The model of the message sequence may be made only with the initial model shown in Figure 1, as it should be taken into consideration a new coordinate: the logical

time of processes.

Lamport [13] proposed a model for the logical time just to be able to correctly present this sequence of events and messages within a distributed system. In his approach, the execution of a process is modelled as a sequence of atomic events, each of them requiring for execution a unit of logical time. In the model of Lamport, sending and receiving messages are considered events within the framework of the processes, as in Figure 9.

Let's consider two correct processes, noted P and Q, who carries out their activity in time. At the level of the process P, the event noted *Send(m)* takes place, by sending a message m to the process Q. At the level of the process Q, two events take place: one is the reception of the message m, *Receive(m)*, and the other one is the delivery of the message m, *Deliver(m)*, to the components of the process Q designed to process the message m.
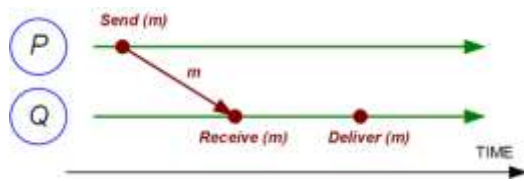

Fig. 9. The evolution in time of the communication between two processes

Modelling the message exchange according to Figure 7 allows the highlighting the sequence of events. For example, at the level of the process Q, firstly the receiving event takes place and subsequently the delivery event of the message to the other components takes place.

The separation between the receiving events and delivery ones of the messages allows the expansion of the abstract model of the system with the protocols for receiving messages, which may consist of a series of activities related to the message before it is delivered to the other components of the process.

Here it should be highlighted a few possible situations:

- Not all messages will be received (network issues could appear);
- Not all received messages must be delivered to the components of the process

(messages could be incorrect or they are sent to another recipient).

Having a model of distributed system in which the execution at the level of processes and communication between processes are presented in connection with the logical time, we can switch to the modelling of the order of the messages based on causality relations.

3.6 The Causal Order of Messages

In the distributed system operations, the processes send several messages to other processes and, in return, receive multiple messages.

Transmitting messages between processes involves the message broadcast over the connection and subsequently the receiving phase and the phase of message delivery to the recipient. Transmitting messages comply with the property called the causal order of messages. This refers to the fact that messages are delivered with respect to the relationship cause-effect. A message must have a property called "happened-before" in order to show this causal order.

Consider now two processes and a message m1 transmitted from a transmitter process P toward a process receiver Q. If, at the level of the process Q, the receipt of message m1 causes the transmission of a different message, noted m2, to a different process R, then a causal order exists between the messages m1 and m2 which is noted as follows:

$$m1 \rightarrow m2 \tag{4}$$

There are several situations in which, in the abstract model of the distributed system, it is considered that the message m1 is a potential cause for the message of the message m2, and these situations are:

1) a process broadcasted the message m1 and, subsequently, the message m2;
2) a process received the message m1 and, subsequently, broadcasted the message m2;
3) at the process level, if a message m' exists, such that m1 exists before m', and m' exists before m2, and these relationships are noted as follows:

$m1 \qquad\qquad \rightarrow \qquad\qquad m'$
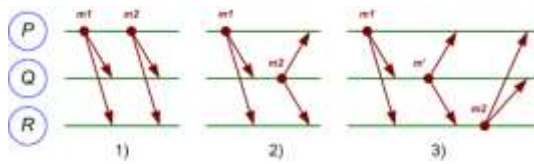and $\quad m' \rightarrow m2$ \hfill (5)

Fig. 10. The causal order of messages (P = transmitter, Q and R = receivers)

In the phase of message delivery to the recipient, the causal order means that if a message is delivered, then all the previous correct messages have been delivered.

In the distributed system model, it can be abstracted the reliable broadcast of messages in which the delivery comply to the causal order of messages as follows:

The properties of causal order are described below:

- *P1. Validity*. If the correct process S broadcasts a message m, then the message is eventually delivered to the destination process R.
- *P2. Non-duplication of messages*. A message is delivered only once.
- *P3. Non-creation of messages*. If a message m broadcasted by the source process S has been received by the destination process R, the message m has surely been issued previously by the process S.
- *P4. Agreement*. If a message m is delivered by a correct process, then the message m is eventually delivered in each correct process in the system.
- *P5. Causal delivery*. If the message m1 is a potential reason for the dissemination of the message m2 (m1→m2), then a correct process is not delivering the message m2 before m1.

4 Centralized Management and Separated Execution

4.1 Assumptions, System Structure, Advantages
Consider now a distributed system in which there are several processes with well-defined roles, located on several hosts, as seen in Figure 9.

The main features of this system are described below:

- The process C, referred to as the client-process, has established a secure connection with the process M, referred to as the manager-process.
- The process M is located on another host than the process C.
- Over the secure connection between these processes, there are messages sent from C to M, called "requests", and also messages from M to C, called "responses".
- The process M receives the request messages sent by the clients and delivers them to the system components with respect of the receiving order.
- The manager-process M is capable of storing the requests made by the client C in a form which permits subsequent finding of a certain request.
- The system also contains the processes W1, W2, W3, referred to as worker-processes, which have each established a secure connection with the manager-process M, but they do not have direct links between them.
- The worker-processes W1, W2, W3 are located on different hosts, other than the host of M.
- The worker-processes W1, W2 and W3 are deterministic, in the sense that when they receive identical messages from the process M, they will carry out the same operations and formulate identical responses which they send back to the manager-process M. Therefore, there are no local factors that could determine different responses to identical requests from the manager-process M.
- As a result of receiving requests from the client-processes, the manager-process M sends the messages to the worker-processes W1, W2 and W3, in the same order.
- The content of the messages sent by the manager-process M to the processes W1, W2, W3 is determined by the two factors below:
  1) The content of the request sent by client-process C;

2) The existence of an agreement on the messages format for the entire system.
- The manager-process M receives the responses sent by the workers W1, W2, W3, and processes these responses in a separate phase of its execution, in order to form a response message for the client.
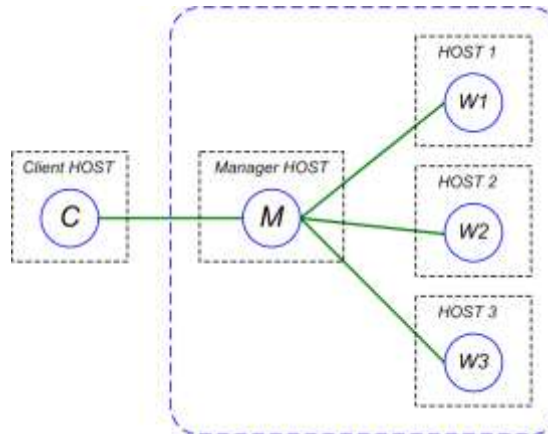

Fig. 11. Distributed system with centralized management and separate execution

The distributed system shown in Figure 11 operates based on the following *assumptions*:
- The connections between the processes are secure, so that the message broadcast is reliable;
- The messages sent by the client-process C are arranged in a FIFO structure at the level of the manager-process M;
- The messages sent by the manager-process M to the worker-processes Wi (i = 1 … n) comply with the causal order;
- The messages received by each worker-process Wi (i = 1, 2 … n) are stored in a FIFO structure and are delivered one by one on all process components, complying with a causal order.

The *advantages* of this system with centralized management and separate execution are below:
- The use of the processing power of multiple computers due to the separate physical location of these processor-intensive processes;
- The location of the stored data is closer to the other systems that may require data access through the use of multiple, separately located, worker-processes;
- Less response time of manager-process M in relationship with the client process C through the separation of management-process and worker-processes;
- Ease of worker-processes debugging, based on their separate location and execution;
- The ability to add, relatively quickly, other workers in the system;
- The ability to use worker-processes W2, W3 … Wn, with the role of *replicas* of a *main process* W1, and having a processing logic identical to the main process W1, and controlled by the same separate manager-process M.

The above-mentioned advantages lead to the idea that such a distributed system as the one in Figure 11, having one centralized management process and several separate execution processes, could be organized as a system providing *secure groups of reliable services* [14]. The algorithms presented below could be used to model such a reliable system.

4.2 The Exchange of Messages between the Client and the Manager
As a result of the user actions, the client-process C sends a message to the distributed system, practically, to the manager-process M, requesting the execution of an action and obtaining a response message. This response message contains data and their volume varies depending on many factors, the most visible being the nature of the requested action.
This exchange of messages between the client and the system requires a prior agreement on the format of the messages, so that the

processes can have a "dialogue".
The message of the client-process contain the name of the requested action (or the requested service) and the parameters required by the

execution.
The procedure which determines the send of a message to the process-manager is:

Execute (action, params, clientID)　　　(6)
This procedure determines the creation of an event at the level of the client-process:
$$\{ C, Send \mid [action, params], manager, self \}　　　(7)$$
The message reaches the manager-process M and as a result, at the level of M several procedures are successively triggered:

    Receive (action, params, client)
    [opID, opName, params] := Operation (action, params)
    Log (opID, opName, parameters)
    Execute (opID, opName, params, worker)
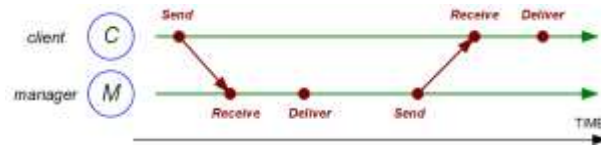   Send (result, client)

Fig. 12. The exchange of messages between the client-process C and the manager-process M

The abstraction of the message exchange between the client-process C and the manager-process M, shown in above figure, as

well as the triggered procedures, are presented in the algorithm below.

ALGORITHM 1.

The messages between Client and Manager, and the specific operations (used notations: *params*=parameters, *opID*=operation ID, *opName*=operation name)

Implements:
    Class Name: Client, instance name: client
    Class Name: Manager, instance name: manager

upon event { manager, Receive | [action, params], client } do
    [opID, opName, params] := Operation (action, params)
    successfullyLog := Log (opID, opName, params);
    if successfullyLog then
        trigger { self, Deliver | [opID, opName, params], client }
    else
        result : = "Logging Error. Abort";
        trigger { manager, Send | client, [opID, result] };

upon event {manager, Deliver | [opID, opName, params], client} do
    workers := CheckWorkers();
    for each worker in workers do
        if (status(worker) == "Ready") then
          trigger { worker, Send | [opID, result] , manager , self };
        else
          trigger { self, Resychronize | worker, opID }

upon event { client, Receive | result, self, manager } do

trigger { self, Deliver | result };

upon event { client, Deliver | result } do
   trigger { self, GetResponse | result };

4.3 The Exchange of Messages between the Manager and the Worker

As a result of a request previously made by a client-process, at the level of the manager-process M, after a request registration, the following procedure is executed:

Execute   (opID, opName, params, worker, self)   (8)

This procedure determines the creation of an event at the level of the manager-process M:

{ M, Send | [opID, opName, params], worker, self }   (9)

As a result, at the level of the worker-process W, these procedures are successively triggered:

   Receive   (opID,   operationName,   parameters, manager)
   Log   (opID,   operationName,   parameters)
   Execute (operationName, parameters)
   Send (opID, result, manager, self)

In the end, the worker-process W sends a message to the manager-process M. As a result of this message, within the manager-process M the following procedures are triggered:

   Receive (operationID, result, worker)
   PrepareResponseForClient (result)

The procedures mentioned above use the type of data called operation. It is actually a public class in the distributed system. Therefore, within the system, there is an agreement for the use of this class, referring to the transport of data between the software components.
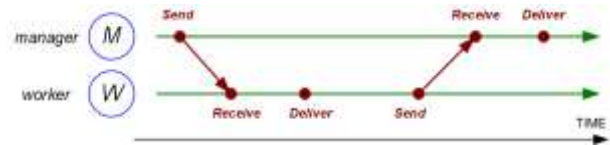

Fig. 13. The exchange of messages between the manager-process M and the worker-process W

The abstraction of the exchange of messages between the manager-process M process and the worker-process W, shown in the above figure, as well as the triggered procedures, are shown in the algorithm below.

ALGORITHM 2.

The messages between Manager and Worker, and the specific operations (used notations: *params*=parameters, *opID*=operation ID, *opName*=operation name).

```
Implements:
    Class Name: Manager, instance name: manager
    Class Name: Worker, instance name: worker

upon event { worker, Receive | [opID, opName, params], manager } do
    successfullyLog := Log ( opID, opName, params );
    if successfullyLog then
        trigger { self, Deliver | [opID, opName, params], manager }
    else
        result : = "Logging Error. Abort";
        trigger { worker, Send | manager, [opID, result] };

upon event { self, Deliver | [opID, opName, params], manager } do
    result := Execute (opName, params);
    trigger { worker, Send | [opID, result] , manager , self };

upon event { manager, Receive | [opID, result], self, worker } do
    trigger { self, Deliver | [opID, result], worker };
```

```
upon event { manager, Deliver | [opID, result], worker } do
    trigger { self, PrepareResponseForClient | result };
```

5 Conclusions

The distributed system proposed in this paper has been analyzed according to specific procedures, on the basis of primordial elements: the process, the connection between processes and the message.

Important aspects concerning the location and the role of the processes in a distributed system have been presented. For the model proposed in this paper, the matter of logical time of process has been presented.

Within the process model, software components and the process levels were abstracted. Subsequently, the communication between the process components based on events has been modelled. There have been analyzed some problems which influence the functioning of a distributed system as a whole, namely, the matter of distributed agreement and the matter of causal order of messages.

A model of distributed system was proposed and analyzed, in which several separate processes have the role of executors of specific operations and the related management execution operations are centralized in a separate process of management. Thus, the specific concerns related to these two categories of operations, management and execution, were separated, according to the separation of concerns principle.

The management and the distinct development of the two categories of processes is considered to be an advantage brought by the proposed system model.

The separation of the management process and the execution processes brings benefits coming from the use of the processing power on multiple computers.

In the model proposed in this paper, the manner of abstraction the communications between processes, based on messages, has been presented. Subsequently, the abstraction of communications between the client and the manager, and also the communications between the manager-process and the worker-

processes was described in details.

The distributed system model and the algorithms presented in this paper could be the starting point for developing a class of distributed systems providing secure groups of reliable services.

References

[1] M. Vida, O. Lupse, V. Gomoi, L. Stoicu-Tivadar, V. Stoicu-Tivadar, E. Bernad, Using Web Services to support the interoperability between healthcare information systems and CDS systems, *Journal of Control Engineering and Applied Informatics*, vol.16, no.1, 2014, pp.106-113.

[2] C. Toader. Multilayer resource-oriented architecture supporting RESTful and non-RESTful resources. *Annals of DAAAM for 2009* and *Proceedings of 20th DAAAM International Symposium*, Vienna, Austria, November 25-28, Vol.20, 2009, pp.467-468.

[3] A. Avizienis, J.C. Laprie, B. Randell, & C. Landwehr. Basic concepts and taxonomy of dependable and secure computing, *IEEE Transactions on Dependable and Secure Computing*, vol.1, no.1, 2004, pp.11-33.

[4] L.E. Moser, P.M. Melliar-Smith, and W. Zhao. Building dependable and Secure Web Services, *Journal of Software*, vol.2, no.1, 2007, pp.14-26.

[5] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme & G. Alonso. Understanding replication in databases and distributed systems, *Proceedings of 20th International Conference on Distributed Computing Systems (ICDCS'2000)*, Taipei, Taiwan, ROC, 2000, pp.464-474.

[6] M.Dahlin, B.B.V. Chandra, L. Gao, A. Nayate, End-to-end WAN service availability, IEEE/ACM Transactions on Networking, Issue 2, 2003, Vol.11, pp.300-313.

[7] M.J. Fischer, N.A. Lynch, M.S. Paterson, Impossibility of distributed consensus with one faulty process, *Journal of the ACM*, Volume 32, Issue 2, 1985, pp.374-382.

[8] H. Yu & A. Vahdat, The costs and limits of availability for replicated services. *Journal of ACM Transactions on Computer Systems*, Vol. 24, Issue 1, 2006, NY, USA, pp.70-113.

[9] J. Vesperman, Essential *CVS*, 2nd Edition, O'Reilly Media Inc., USA, 2006.

[10] F. Pedone, Boosting system performance with optimistic distributed protocols, *IEEE Computer*, Vol. 34, Issue 7, 2001, USA, 80–86.

[11] Y. Saito, & M. Shapiro. Optimistic replication, *ACM Computing Surveys*, vol.37, nr.1, 2005, pp.42-81.

[12] C. Toader. Increasing reliability of Web services. *Journal of Control Engineering and Applied Informatics*, Vol.12, No.2, 2010, pp.30–35.

[13] L. Lamport, Time, Clocks, and the Ordering of Events in a Distributed System, *Communications of the ACM*, Vol.21, Issue 7, 1978, pp.558-565.

[14] C. Toader, C. Rădulescu, C. Anghel, G. Boca. Organizing Secure Groups of Reliable Services in Distributed Systems, *Proceedings of the 14th International Conference on Informatics in Economy (IE 2015)*, University of Economic Studies ASE Bucharest, Romania, April 30 – May 03, 2015, pp.44-49.

Cezar TOADER graduated the Faculty of Sciences at Technical University of Cluj-Napoca, with a diploma in Economics. He holds a PhD diploma in Computer Science from 2011, obtained at "Politehnica" University of Timișoara. He had gone through all teaching positions since 1992 when he joined the academic staff. Currently he is Professor within the Department of Economics, Faculty of Sciences, at Technical University of Cluj-Napoca. He is the author/coauthor of many books and scientific articles in his fields of interest: Architecture and reliability of distributed systems, Web programming, Web services.

Rita TOADER graduated the Faculty of Sciences at Technical University of Cluj-Napoca, with a diploma in Economics. She had teaching positions since 1992, and now she is Associate Professor PhD within the Department of Economics, Technical University of Cluj-Napoca, and also the Vice-Dean of the Faculty of Sciences. She is the author/coauthor of scientific articles and books related to Information systems and Modelling of economic processes.

www.manaraa.com