THE PROBLEM OF TIME IN SPECIFYING INPUT DATA FOR THE CORRELATION LAYER FROM THE MODEL OF EMERGENCY WARNING SYSTEMS

Nicolae-Dorel CONSTANTINESCU[1]

Aceast articol tratează problema marcajului temporal al alertelor în sistemele de avertizare pentru situatii de urgență. Pentru o bună corelare, trebuie să se cunoască corect informația temporală a alertelor. O serie de tehnici sunt considerate şi se prezintă o soluție bazată pe implementarea ştampilelor de timp Lamport, în concordanță cu standardele de alertare existente.

This article addresses the issue of temporal markup of the alerts in the Emergency Warning Systems. For a correct correlation, must know the accurate time information of alerts. A number of techniques are considered and a solution is presented, based on the implementation of Lamport timestamps, in accordance with existing alerting standards.

Keywords: alerting, warning, alerting systems, warning systems, order, time, Lamport, emergerncy warning systems (EWS)

1. Introduction

The problem of time is perhaps the most complex problem that the human thinking tries to comprehend and to solve. It is being studied by many disciplines in the field of knowledge, from the humanistic sciences, like philosophy, continuing with the natural sciences, like physics, and ending with today's areas of technological development, such as computer science.

Recently defined as "a persistently stubborn illusion" (Einstein) or "an abstraction at which we arrive through the changes of things" (Mach), so as a measure of change, has been shown that the concept of time makes sense only when intrinsically linked with the notion of space. Contemporary technological advances, such as launching GPS satellites on orbit, validated in practice the theoretical discoveries made by Einstein in the early 1900s. However, it's a hard challenge for current thinking, which is the one according to, today's systems were and are still being designed, not to rely on the human perception of time, closer to what Isaac Newton postulated in 1689 (and subsequent proved

[1] PhD student, Dept. of Automatic Control, University POLITEHNICA of Bucharest, Romania, e-mail: dorel_nic@yahoo.com

incorrect), namely the absolute time, true and mathematical, of itself and from its own nature, flowing equably without relation to anything external.

In a previous paper we introduced, in the model of Emergency Warning Systems (EWS), the so-called Correlation Layer [1]. The correlation may take place within a system, at the sensor network level, or in the aggregate of warning systems, so that they can synchronize alerts and create prerequisites for the seamless functioning as a system of warning systems. Contributions of this article are the motivation of study along with the specific formulation for the problem of time of alerts in EWS, the evaluation of different ways to solve the problem, the proposal to complement the standards for emergency warning with a *logical clock* element and the implementation of an effective software solution. This article is further organized as follows: in section 2 considerations about related work are given, in section 3 the temporal dimension of alerts in warning systems is outlined, paragraph 4 presents a number of approaches to the problem of time in distributed systems and extracts a solution for alerting systems for emergency, paragraph 5 briefly describes the application implemented, in paragraph 6 a test for the application, on an alerting scenario, is made, paragraph 7 presents the *matrixClock* element proposed for inclusion in alerting standards and paragraph 8 draws the conclusions and outlines the ways to go forward.

2. Related work

Time is a decisive issue in case of emergencies, so the problem of time naturally arises in the study of EWS. Besides, time and timing are important in any computer system and, more so, in any distributed system, and EWS includes features of both mentioned systems. Problems like congestion, communication links failure, human response or other difficult conditions bring technical challenges for systems used in emergency management. On one hand, considerations regarding time may refer to the available warning time provided by warning systems. That time window depends on the dynamics of events [2] and gives people the change to take preventive actions. For example, the warning time that can be obtained by an earthquake warning system in Bucharest is 25-30 s maximum [3]. At the other end there are warning systems for slow-onset events, such as global warming, where several years are available to take preventive actions. On the other hand, time, as well as place of occurrence represents basic information about emergency events, so it has to be manipulated in a suitable manner, by distributed, real-time and geographical (GIS) systems and databases. In [4], valuable considerations about real time databases, including scheduling, resource reservation or data consistency issues are given. The dynamic of events is considered in [5], in context of geographical information systems, emphasising on real-time data, as it enable emergency operators to accurately act in

emergencies. Providing solutions for the problem of time in EWS is somehow a new challenge and we base our work on knowledge of distributed systems, a field well covered with rich literature, for example [6].

3. The order and time of the events in the correlation process context

In order to perform its tasks, the correlation system must, among others, know the correct order of events. When an event detected by a sensor X happens before another event detected by a sensor Y, it is necessary that time signatures for the corresponding alert messages, which are sent to the correlation system, to reflect that order. Because the system where messages are exchanged is a distributed one, where messages originate from various sources, a new problem that arises it to correctly establish the temporal precedence of events. It is possible for a sensor X to associate an event A a timestamp later than the one that a sensor Y associates to another event B, however the event A to occur before event B (perhaps simply because the clock of X is later than the clock of Y). Among the obvious solutions, one can state synchronization with precise time servers and the use of the time moment immediately before the emission of alert, as reference time. But let's not forget that in discussion here are emergencies, when problems like delay, delay variation or network congestion must be paid attention. It can happen for some messages to arrive late, others to arrive out-of-order and others to not arrive at all, which would cause real problems for the correlation phase. Time problem in distributed systems is not new and has been intensively studied. Regarding warning systems for emergencies and, in particular, the correlation of alerts, certain issues should be taken into account before attempting to formulate a solution. Thus, for the correlation process it is less important to know the standard time an event occurred, but of major importance is the order of events' occurrence. The correlation process, implicitly the ordering of events, should also perform in situations of network congestion, involving a minimum required communication resources, especially in critical moments. At the same time, the idea of a central time server does not seem very good, having in mind its availability and integrity during the crisis.

4. The problem of finding the time of alerts in emergency warning systems

4.1 General context of the time problem

The problem of determining the time of the alerts in warning systems for emergencies is an application or a particular case of the problem of determining the time in distributed systems.

The first issue that needs to be discussed is on the nature of clocks used. Thus, a boundary line is given by the choice between using physical clocks and

using logical clocks. Physical clocks are the ones well known (with further improvements), which measure the Newtonian time, and logical clocks, introduced by Lamport, are numbers related to events taking place in various processes of the system, where the numbers are interpreted as moments in which the events occurred. More specifically, for every process *Pi,* a clock *Ci* is defined, as a function that associates a number *Ci(a)* to every event *a* of that process. If an event *a* occurs before another event *b*, then it must happen at an earlier *time* then *b.* This condition is called *The clock condition* and formally is written as follows: if *a* occurs before *b*, than: *Ci(a)* < *Ci(b)*. For the system of clocks to meet the condition, the processes must obey the following three implementation rules: (1) Every process *Pi* increments its *Ci* between any two successive events; (2.a) If an event *a* is the sending of a message *m* by process *Pi*, then the message *m* contains the timestamp *Tm = Ci(a)*. (2.b) After receiving a message *m*, the process *Pi* sets *Ci* as the maximum of current value and *Tm* incremented by one unit [7].

A second nuance is related to the physical principles given by the theory of relativity. It is shown that time and space are inextricably linked; *how* time passes is linked to *where* we relate. The systems in discussion, those for emergency warning, are located on Earth, a geoid of rotation around its axis, orbiting the Sun and having gravitational field. The systems are connected by extended networks with variable delays. Speed and gravity affects the time passing. The differences are really small and do not require frequency adjustment mechanisms (as in a satellite launched into orbit), but a cautious, "safe" approach to the problem of time, which takes into account, and minimizes the influence of these phenomena, is welcomed.

Having assessed the type of clock and the laws by which it works, another problem is the synchronization and keeping synchronized the system of clocks. Clocks synchronization problem in distributed systems is not new. It was shown that in a distributed system with *n* processes connected by a network topology, communicating through messages, clocks cannot be synchronized with an accuracy better than $\varepsilon(1-\frac{1}{n})$, where is ε the uncertainty in message delivery time delay or the inverse of the difference between minimum and maximum possible delay [8]. The above result specifies, in probabilistic terms, a lower limit for how close the indications of two clocks, subjects to a synchronization algorithm, can be, at the same moment of real time. Another result provides the maximum degree of simultaneity that can be achieved in a network, namely the difference in real time at which the two processes reach the same value of their clocks. This is more than half of the uncertainty in the delay of messaging transfer [9]. It was also shown that the synchronization between processes can be achieved only if less than one third of them work badly (in the sense of Byzantine faults, when system components fail in arbitrary ways, not just by crashing, but by

incorrect processing of requests, the corruption of local state, producing inconsistent and incoherent results).

Thus, in terms of timing, let's initially note the possibility of internal and external synchronization of systems' clocks. A set of clocks is internally synchronized if at each point of real time the distance between the actual values of two correct clocks is bounded by an a priori specified constant called maximum internal deviation and each clock is operating on a linear envelope of real time (maintaining certain uniformity). A clock is externally synchronized if at each point of real time the difference between its value and the actual time is bounded by an a priori specified constant called maximum external deviation [10].

Clock synchronization can be achieved globally, for the entire system, or only locally, for some regions. For economic reasons, sometimes the second option may be preferred.

When the synchronization method ensures the bounding of deviation in a certain domain, it is called deterministic, and when there is a not null probability (possibly determined as value or limits) that the deviation exceeds a certain threshold, the synchronization method is called stochastic.

Synchronization can be achieved before the occurrence of events, which is called a priori synchronization or later, after the events were registered, making a posteriori synchronization.

4.2 Particular requirements. Design decisions. Extracting a solution

As mentioned above, what we pursue is an alerting process of high quality. Poor warning and, consequently, failure of it, is translated by high costs paid by society. Alert correlation process aims to adjust certain parameters from the expression of the cost function, intending to decrease its value. Further, for a good correlation, it is necessary for the temporal dimension of alerts to be known, as accurately as possible. In addition, warning activity itself must be done in timely manner.

A solution to the problem detailed above, in the context of emergency alerting, has to meet several conditions that cumulatively complement requirements for solving the time problem: compatibility with currently existing alerting standards, compatibility with communication protocols, compatibility with a wide range of hardware and software running on top of it. Prior to formulating a solution, certain design decisions have to be taken. Thus, the participating nodes in the process of alerting are supposed to be computer machines, prepared for networking, by existing protocols. The layer at which the solution is implemented (from the OSI reference model) is Application Layer, and the language used for implementation is Java, because it is supported on a wide range of hardware platforms. The solution is based on the use of Lamport clocks, with the additions that a node stores its knowledge about the values of clocks for

the whole system (vector timestamps), as well as knowledge about the values of other vector timestamps, as they are known to be retained by other nodes in the system (resulting in matrix timestamps). An event (alert in our case) is associated a particular timestamp on which one can determine the order of occurrence.

We believe that this is the right approach, one that correctly opens the way for solving the problem in question, for several reasons:

- the solution is decentralized, does not require certain main components, resulting in better scalability;
- the agents are similar units that can be standardized;
- the solution is consistent with the alerting standards;
- the solution is consistent with the "cautious approach" of the time, regarding the principles of the theory of relativity;
- implementation is built on open software technology and supported by a wide range of hardware platforms.

5. Application structure

The application is built from several classes, as follows:

AlertNode - the class where the node's matrix logical clock is instantiated and which contains the function *main()* that launches the client and server threads. At program launch we specify into the command line the node identifier, the total number of nodes in the system and port that the server will listen on.

AlertServerThread - class that implements the server capable to receive alerts, by a communication protocol established with the alert clients. For each connection it creates a dedicated thread, so it can serve many customers simultaneously.

AlertProtocol - is class that implements the communication protocol between the alerting server and client. This class contains *readAlert* method through which *CAPHandler* class is instantiated, and further, with its help, the XML Common Alerting Protocol (CAP) alert, sent by the client, is parsed. Also, when the alert is received, the *receiveAction* method of the matrix clock is called, which implements the logic applied to clock upon receipt of an alert.

CAPHandler - is the class that handles parsing CAP alert sent to the server.

MatrixClock - is class that implements the matrix logical clock.

AlertClientThread - is the class through which alerts are sent to the server. After a short initial handshaking, the client was invited by the pair server thread to send the alert. This one puts it on line and if the alert is received correctly, a confirmation message, containing the identifier of the last received alert, is sent, along with an invitation to send a new alert; if the alert was not received properly

(condition currently achieved only if the console was introduced other message that the one requested by the protocol), server requests alert's retransmission. The client transmits only CAP formed alerts.

6. Showing an operating example

Consider the alert scenario in the next figure. Ellipses represent alerting network nodes, each with a unique identifier. The rectangle above nodes is the port number on which it listens. The arrows represent the direction and way of transmission of alerts, and the numbers on them, the sequence in which alerts are sent. On each of these nodes a software agent is running, having the dual function of client and server for alerting. Alerting is done in connection oriented mode, using TCP stream sockets. A socket is uniquely identified by an IP address (node address) and port (through which data is directed to their intended application). Each server waits for connections on the specified port. In order to do testing on a single machine, we consider nodes to differentiate by port that the server listens on.

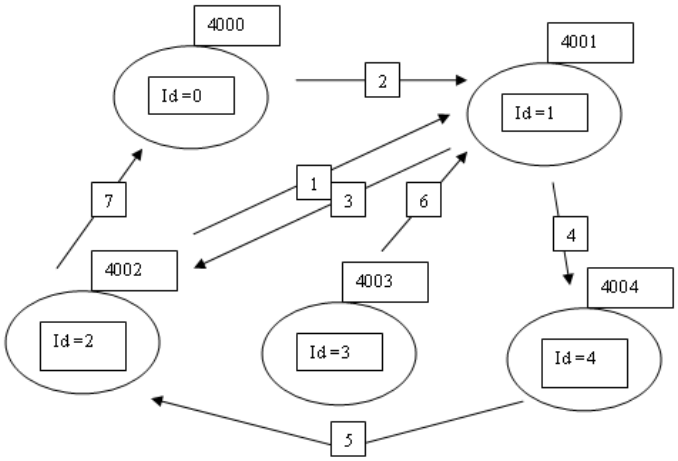

Fig. 2. Alerting scenario

When a node has an alert to send to another node, on the client thread, the application tries to connect to destination node. When the server receives the connection request, it fires a separate thread that deals with the request and receives the alert, all of that while remaining in a state of listening and capable to accept other connections. As to the transmitter, in the current form of the solution, it can send a single alert at a time. Launching the client is done form the command line, by writing at the console the address and port to which will send the alert, then the messages corresponding to the established alerting protocol. Alert generation is done automatically, focusing on the issue of interest here, namely

the timestamp. Additionally, a unique alert identifier is generated, identifier composed of the identifier of the node and another identifier unique on the machine (GUID). Warning message is encoded in CAP (Common Alerting Protocol) format. In a production version, the client must wait no longer for commands from the user, it can periodically query a database in which the sensors put their records, and based on internal logic it can decide what nodes should be alerted. In the figure below one can see the values of the matrix clocks of nodes in the system, at different times, according to the first steps of the alert scenario described above. At the end of the process, the clocks have the values next to the heads of the arrows.

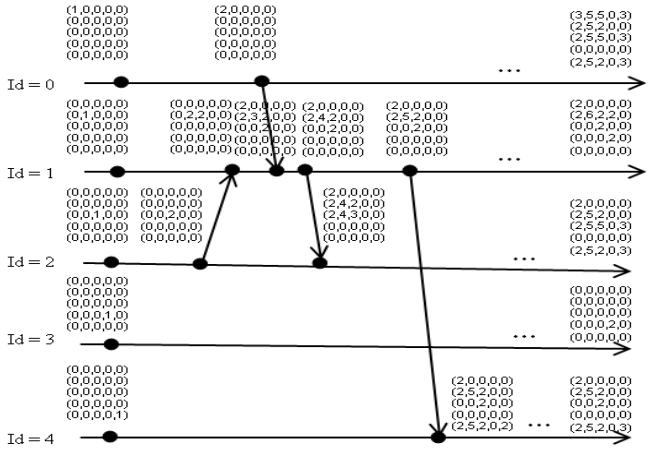

Fig. 3. the flow of alerts and the matrix clocks of the nodes

In the window below, one can see the client window and examples of the messages exchanged in the system:

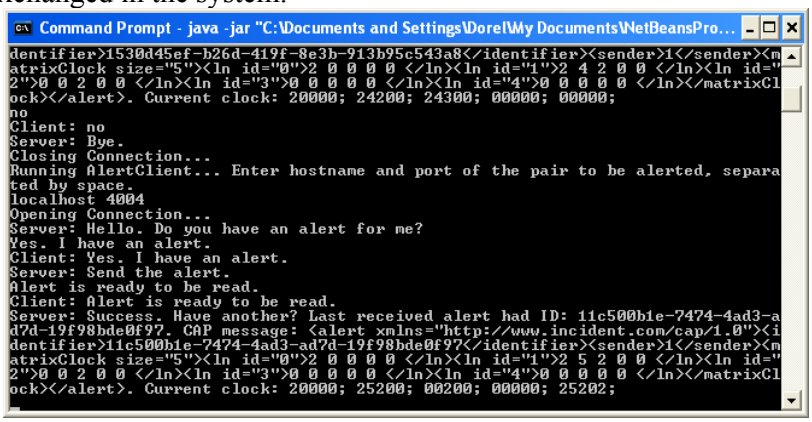

Fig. 4. Client console

The server sends back the client the request message received from it and the message resulting from processing that request. If the alert was received successfully, the latter is composed of the identifier of the last received alert, the CAP message that carried it and the current matrix clock. In a production version, only the identifier may be kept, in order to make confirmation of alert receipt. Serverul trimite înapoi clientului mesajul cerere primit de la acesta şi mesajul rezultat în urma procesării cererii.

7. Complementing the standards with matrixclock element

The proposed solution introduces into the structure of alerting messages the element *matrixClock*.

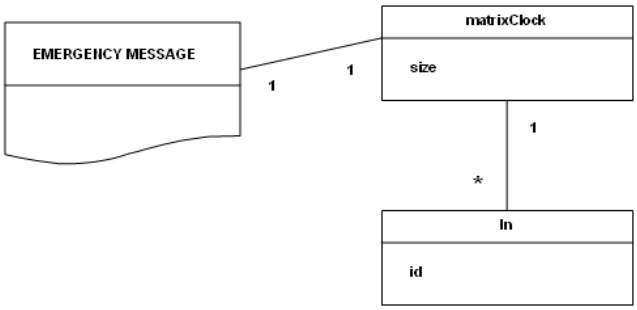

Fig. 5. The DOM of the element matrixClock

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="matrixClock">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="ln"/>
    </xs:sequence>
    <xs:attribute name="size" use="required" type="xs:integer"/>
  </xs:complexType>
</xs:element>
<xs:element name="ln">
  <xs:complexType mixed="true">
    <xs:attribute name="id" use="required" type="xs:integer"/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Fig. 6. XML Schema of the element matrixClock

8. Conclusion and further development

In this article we motivated by a simple cost analysis the need to correlate alerts in warning systems for emergencies, we deduced the necessity of knowing the correct order of alerts and we implemented a solution for this, based on an Lamport type algorithm, introducing the element matrix clock in the structure of alerting protocols.

Further additions, designed to enhance the effectiveness of the mechanism, may be made. We can introduce a fault detection component against the Byzantine errors (which will be located in the algorithm at a message receipt). Thus, knowing that a node *n* was aware of the system's timestamps, can additionally verify at the receiver that it had properly assigned timestamp (e.g. by checking that each secondary vector of the matrix is smaller than the main vector – the one corresponding to the current node) . It is correct that after a period a node has not responded, to invalidate the corresponding line in the matrix (since it is unlikely to be up to date), using an additional column in the matrix named *valid*. Each alert message can be signed with the private key of the transmitter and at the receiver to verify its authenticity with the public key. It might also, adaptively to frequency at which nodes exchange messages, if within a specified interval a process did not make send/receive operations, to exchange specific messages exclusively serving to update the clock.

R E F E R E N C E S

[1] *N.-D. Constantinescu,* It Supporting Delivery of Input Parameters for the Correlation Layer of the Multi-Layered Emergency Warning Systems Model, Scientific Bulletin of Politehnica University of Bucharest, Series C, **Vol. 73**, Iss. 3, 2011

[2] *V.F. Grasso,* Early Warning Systems: State-of-Art Analysis and Future Directions, United Nations Environment Programme (UNEP), http://na.unep.net/geas /docs/Early_Warning_System_Report.pdf

[3] *F. Wenzel,* An Earthquake Early Warning System for the Romanian Capital, Perspectives in Modern Seismology, Lecture Notes in Earth Sciences, **vol. 105**, p.1

[4] *J. A. Stankovic,* Misconceptions About Real-Time Databases, *IEEE Computer Journal,* **Vol. 32**, Pages 29-36, 1998

[5] *M.-P. Kwan, J. Lee,* Emergency response after 9/11: the potential of real-time 3D GIS for quick emergency response in micro-spatial environments, Computers, Environment and Urban Systems Jounral, **Vol. 29**, pg. 93–113, 2005

[6] *A.S. Tanenbaum, M. van Steen*, Distributed systems: principles and paradigms, Pearson Prentice Hall, 2007

[7] *L. Lamport,* Time, Clocks, and the Ordering of Events in a Distributed System, Communications of the ACM, **Vol. 21**, Number 7, July 1978

[8] *J. Lundelius, N. Lynch,* An upper and lower bound for clock synchronization, MIT Information and Control Journal, 62, 190-204, 1984

[9] *D. Dolev, I. Halpern, J. Strong , H. Raymond*, On the possibility and Impossibility of achieving Clock Synchronization, *Journal of Computer and System Sciences*, **Vol. 32**, Number 2, April 1986

[10] *C. Fetzer,* Integrating External and Internal Clock Synchronization, *Real-Time Systems*, **Vol. 12**, Number 2, 123–171 (1997)

[11] *** Distributed Systems Course Slides, Texas University, http://users.ece.utexas.edu/~garg/dist/jbk-slides

[12] *** *P. Borrill*, Rethinking Time in Distributed Systems Can We Build Complex Systems Simply?, online presentation by Stanford Edu