Artículo de Investigación. Revista Ciencia e Ingeniería. Vol. 36, No. 2 pp. 73-84, abril-julio, 2015.
ISSN 1316-7081. ISSN Elect. 2244-8780 Universidad de los Andes (ULA)

Crónicas Distribuidas para el Reconocimiento de Fallas

Distributed Chronicles to Faults Recognition

Vizcarrondo Juan[1*], Aguilar José[2], Exposito Ernesto[3] and Subias Audine[4]
[1] Centro Nacional de Desarrollo e Investigación en Tecnologías Libres (CENDITEL), Mérida – Venezuela
[2] CEMISID, Dpto. de Computación, Facultad de Ingeniería, Universidad de Los Andes, Mérida – Venezuela
Investigador Prometeo, Universidad Técnica Particular de Loja, Loja, Ecuador
[3] CNRS, LAAS, 7, avenue du Colonel Roche, F-31400, Univ de Toulouse, INSA, LAAS, F-31400, Toulouse - France
[4] CNRS, LAAS, 7, avenue du Colonel Roche, F-31400, Univ de Toulouse, INSA, LAAS, F-31400, Toulouse - France
*jvizcarrondo@cenditel.gob.ve

Resumen

El paradigma de crónicas ha sido utilizado para determinar fallas en sistemas dinámicos, permitiendo modelar las relaciones temporales entre eventos observables y describir los patrones de comportamiento del sistema. Los mecanismos utilizados hasta ahora suelen utilizar métodos semi-centralizados, que consisten en un componente central, que es responsable de hacer la inferencia final sobre el diagnóstico de fallos del sistema, basado en la información obtenida de los diagnósticos locales. Este modelo tiene problemas cuando se implementa para el monitoreo de sistemas muy grandes. En este trabajo se propone un sistema de diagnóstico de fallos basado en crónicas distribuidas.

Palabras clave: Reconocimiento patrones distribuidos, reconocimiento patrones temporales, crónicas, diagnóstico distribuido de faltas, composición de servicios web tolerantes a fallas.

Abstract

The chronicles paradigm has been used to determine fault in dynamic systems, allow modeling the temporal relationships between observable events and enabling to describe the patterns of behavior of the system. The mechanisms used until now usually use semi-centralized approaches, which consist of a central component that is responsible for making the final inference about the fault diagnosis of the system, based on the information collected from local diagnosers. Hence this model is not suited for monitoring very large systems. We propose in this article a fully distributed approach. This distributed chronicle recognition is illustrated in the context of e-commerce with a Service Oriented Application implementation

Key words: Distributed pattern recognition, temporal patterns recognition, chronicles, distributed fault diagnostic, web service composition fault tolerance.

1 Introducción

In general, fault diagnosis mechanisms require formalisms for recognizing fault situations. One of the formalism is the chronicles, which allow modeling the temporal relationships between observable events in a system, enabling to describe the patterns of behavior of the system. Dousson showed in earlier work on chronicles as a set of patterns, each characterized by observable events and temporal constraints among themselves and with respect to the context, representing an interpretation of what is happening in the dynamics of the system under study at a given time (Guerraz y col., 2004). Thus, each chronicle represents a situation or scenario of normal or abnormal performance of the system.

The chronicles have been used in different scenarios (Le Guillou y col., 2008), but especially in the analysis of alarms in the supervision for telecommunication network, intrusion detection, voltage distribution network and web services composition. Typically, diagnosis is performed by a central diagnoser responsible for analyzing the set of events generated of the all that are part of the system, when the size of the system increases the problem becomes unmanageable to increase dramatically the frequency of events and the computational complexity of inference.

In previous work, we proposed a distributed architecture for fault diagnosis in service composition, in which the fault diagnosis is performed through the interaction of diagnosers present in each service (Vizcarrondo y col., 2012). To complete this architecture, this paper proposes an extension to the formalism of chronicles, as mechanisms for modeling distributed failure patterns. Chronicles are decomposed into sub-chronicles linked by special events. Moreover, the architecture will be a mechanism for chronicles recognition fully distributed.

This paper is organized as follows, first we present the previous works in the area, second we present our distributed architecture for fault diagnosis proposed in (Vizcarrondo y col., 2012). Then we present our extensions to the formalism of chronicles, culminating with an example of use in service-oriented applications.

2 Related Works

In the fault diagnosis of dynamic systems, some studies have used the formalism of chronicles to determine the faults present in the system (Cordier y col., 2000, Cordier y col., 2007, Quiniou y col., 2001, WS-Diamond 2008), commonly using a centralized or global diagnoser. On the other hand, there is not studies about distributed scheme of fault diagnosis using chronicles. Additionally, (Guerraz y col., 2004) studied the chronicles using petri nets, enriching the chronicles with pre and post information about events condition. Also, (Mhalla y col., 2010) propose another mechanism for distributed failure diagnosis using chronicles, for which the chronicles are decomposed into many sub-chronicles as components are in the system, and observations are communicated between diagnosers, to thereby obtain the necessary information that is not available locally.

Furthermore, (Aghasaryan y col., 1998, Grosclaude, 2004) construct patterns used by the diagnostician like a puzzle, allowing the distribution of the input events in the paths followed by the components for identifying occurrences of faults. All transitions are transformed as pieces describing the partial state transition when are instantiated (turning on the pre and post conditions of the events), allowing to track the occurrence of set of event for the recognition of faults. Both approaches are based on the prediction of missing events using stochastic Petri nets.

Another chronicle approach used in service composition is presented in (WS-Diamond, 2008). This architecture is based on a decentralized diagnosis, where local diagnosers working with a global coordinator, reporting events (called brokering events) from all local diagnosers, using colored distributed chronicle, to global diagnoser, for the recognition of the global chronicle. (Cordier y col., 2007) adapt the chronicle-based approach to a distributed context. For that, they propose a decentralized architecture and an algorithm which is in charge of synchronizing the local diagnoses, and merging them into a global diagnosis. On the other hand, (Boufaied y col., 2004) proposes a distributed check constraints using local and global events, which is enhanced by introducing delay between the communications present in the different diagnosers. This architecture is fully distributed, the global events are spread among different diagnosers. When a local diagnoser recognizes its local chronicle, then propagates global events to the rest of the local diagnosers for making the (complete chronicle) global diagnosis.

There is other set of works about distributed diagnostic for industrial processes (continuous systems). For example, (Roychoudhury y col., 2009) propose an online, distributed, model-based diagnosis scheme for isolating abrupt faults in large continuous systems to overcome the problems on the centralized diagnosis approaches (memory and communication requirements, scale poorly, etc.). (Boel y col., 2002) propose an algorithm for decentralized failure diagnosis with asymmetric communication in which each diagnoser sends only that subset of failure states which is relevant for the other diagnosers. (Le Mortellec y col., 2013) propose a holonic cooperative fault diagnosis approach, to increase the embedded diagnosis capabilities of complex transportation systems. This concept is applied to the fault diagnosis of door systems of a railway transportation system. Finally, (Nakata y col., 2013) propose an approach to detect faults, even if at most n−k local diagnosis decisions are not available, by using the remaining diagnosis decisions.

Recently, in (Vizcarrondo y col., 2012) we proposed a reflective middleware architecture for fault management in service composition, in which each service is supervised by a local diagnoser using chronicles. To complete the proposal, this paper proposes the fault diagnosis system in web service composition, based on a chronicle recognition mechanism fully distributed.

The main difference of our approach with previous works is that we propose a fully distributed model of chronicles, which can be used by distributed systems (like the distributed diagnosers). The previous works based on chronicles are decentralized approaches. Finally, the previous works on distributed diagnoser approaches have not been defined for distributed applications.

3 Chronicles

A chronicle is a set of events with time constraints between them, which represents an interpretation of what is happening in the dynamics of the system under study at a given time (Dousson 2002). Each chronicle situation or scenario represents a normal or abnormal performance of the system, which can be seen as a pattern of behavior of the system in this situation. It is composed of a group of observable events, temporarily restricted by time of occurrence. A chronicle could generate new events and actions at the time of recognition of their occurrence, which could be used as inputs for other chronicles (is a process of inference between chronicles (WS-Diamond 2008). Thus, in (Le Guillou y col., 2008) defines a chronicle C as a "pair (E, T), where E is the set of events and T a set of constraints between their times of occurrence. When their variables and times of occurrence are instantiated, it is called an instance of the chronicle".

An event in the chronicles defines what is observed in the system at a given instant of time, and can be described in different ways (Le Guillou y col., 2008):

1. The name of the event/activity observed (act).
2. The name of the event/activity enriched by the fact that the activity is beginning (act^-) or ending (act^+).
3. The name of the activity enriched with some parameters (variables) that must be observed when event/activity happens (event(?Var_1, ...,?Var_n)).
4. A combination of 2 and 3:

act^-(?Var_1, ...,?Var_n) "act is starting with the parameters ?Var_1,. . .?var_n ".
act^+(?Var_1, ...,?Var_n) "act is ending with the return values ?Var_1,. . .?var_n".

In the chronicles is necessary to define the pair (E,?T), where E is the event name (as described in some of the ways mentioned above) and T the time of occurrence of the event. As has been said previously, an instantiated event is an event in which variables and their time of occurrence have been instantiated.

A tool for chronicles recognizing, called CRS (Chronicle Recognition System), has been developed by Dousson (Dousson y col.,., 1993, Dousson 2002). It is responsible for analyzing the flow of events and recognizing, in real time, any pattern matching with a situation described by a chronicle. When a new event is logged in the system, new instances of chronicles are generated in the set of hypotheses.

The implementation of chronicles using centralized mechanisms in systems containing a large number of components, it is costly in terms of design and computational resources, therefore, the use of distributed recognition mechanisms is suitable to achieve the diagnosis of failures in these systems. In the next section, we present our extension of paradigm of Chronicles to address the distribution aspects.

Reified temporal logic formalism represents an efficient approach to enter the time with the logic, to reach a precise analysis and logical formulation of humans temporal activity. This formalism can be used on the representation of chronicles, allowing propositional terms with temporal objects (reifying predicates).

A temporal proposition is represented as a tuple A(a1, ..., an): V, where A is an attribute name, a1, ..., an are its arguments, and V are the values of the arguments.

The Reifying predicates used in chronicles are (Morin y col., 2003):

- **hold:** represent persistence of the value of a atemporal propositions over a time interval. hold A : (V), (t_1, t_2)).
- **event:** denote a change of the value of atemporal proposition. It has not duration and expresses a time stamped of a pattern. event(A: (V), t).
- **noevent:** used to express the absence of events in a time interval in a chronicle. noevent(A, (t_1, t_2)).
- **occurs:** times between the two time points t_1 and t_2 . The value ∞ can be used for n_2. occurs((n_1, n_2), P, (t_1, t_2)), where ($0 \leq n_1 \leq n_2$).
- Thus, the representation of a chronicle is carried out by specifying (Morin y col., 2003):
- A set of time points.
- A set of constraints between time points.
- A set of atemporal propositions representing activities that occur in the chronicle.
- A set of Reifying predicates representing the context of the occurrences of atemporal propositions.
- A set of external actions to execute when the chronicle is recognized.

Then chronicle model can be written as:

Chronicle Model {
 Events{
  event(e1, T1), event(e2, T2), event(e3, T3) **}**
 Constraints{
  T2-T1 < C1
  T3-T2 < C2 **}**
 When recognized{
  action1
  action2 **} }**

Where:

- e_i ($\forall$ i=1, …, 3) represents the set of atemporal propositions representing activities (events) in the chronicle.
- T_i is the time points of occurrence of the events.
- Constrains: They are the set of constraints between T_i.
- C_i are constants representing the difference among the time points of occurrence of two events.
- Actions: are the set of actions to execute when the chronicle is recognized.

4 Our extension to the paradigm of Chronicles: Distributed Chronicles

4.1 Definition of distributed chronicles

To begin to define our extension, it is essential to clarify that the detection of distributed chronicles is carried out by detecting a set of events E={E1, …, E2, ..., Ep}, distributed among the different n processes of the system (Boufaied y col., 2004). In general, such events can be grouped into n sub-chronicles distributed. The recognition of the n sub-chronicles results in the recognition of a full chronicle. So, we can say that:

Definition 1: "A chronicle can be decomposed into n-sub-chronicles, in which each sub-chronicle SCi is assigned to a site/component Pi of the system under study, and describes a sub-set of events Eaci, with Taci temporal restrictions, that must occur in that site i in order to occur the chronicle recognition".

$C(E,T) = UNION_{i=1,n}(SC_i(Eac_i, Tac_i))$ (1)

where,

- Eac_i and Tac_i correspond to a set of events and temporal restrictions of the chronicle assigned to each component i, where $Eac_i = \{E_k,..., E_l\}$, $Tac_i = \{T_k,..., T_l\}$ and $E_{k, ..., } E_l \in E$, $T_k , …, T_l \in T$ and these events E_k, …, E_l occurs in site i}.

- UNION is a predicate defined by the union of set of events (Eac_i) and of set of temporal constraints (Tac_i) distributed in the n sub-chronicles.

To prove this definition, suppose:

$$C(E,T) = UNIONj=1,p(Ej,Tj) \quad (2)$$

$$C(E,T) = \{(E_1, T_1), (E_2,T_2) ....(E_p, T_p))\} \quad (3)$$

Now, suppose that we distribute these p events among n components, such that p ≥ n:

$$C(E,T) = \{(Eac_1, Tac_1), (Eac_2, Tac_2) ... (Eac_n, Tac_n)\} \quad (4)$$

Then, we can say that:

$$C(E, T) = \{SC_1(Eac_1, Tac_1), SC_2(Eac_2, Taci_2), \dots, SC_n(Eac_n, Tac_n)\} = UNION_{i=1, n}(SC_i(Eac_i, Tac_i)) \quad (5)$$

Definition 2: because a chronicle C can be decomposed into n sub-chronicles (SC), the recognition of the global chronicle can be carried out in a sub-chronicle SCi, recognizing its events (Eaci, Taci) with the union of the partial recognition of the other sub-chronicles (SCj ∀ j=1,n | j≠i) of the other events (in this case, the other sub-chronicles must send it a message to inform it the recognition of their events). In this way the sub-chronicle SCi recognition the chronicle C(E, T):

$$C(E, T) = \{(Eac_i, Tac_i), UNION_{j=1, n \mid j \neq i}(SC_j(Eac_j, Tac_j))\}(6)$$

To prove equation (6), Chronicle can be decomposed into n SCs (equation 1):

$$C(E, T) = \{SC1(Eac1, Tac1), SC2(Eac2, Tac2), \dots, SCn(Eacn, Tacn)\} \quad (7)$$

due to the recognition of the chronicle C is carried out in site i, that is equivalent to:

$$C(E, T) = \{SCi(Eaci, Taci), UNIONj=1, n \mid j \neq i(SCj(Eacj, Tacj))\} \quad (8)$$

Particularly, because a chronicle is decomposed into n sub-chronicles, it is necessary to extend the representation of sub-chronicles to correlate the events recognized between different sub-chronicles, leading then to the recognition of the global chronicle. For this, we extend the formalism of chronicles to manage the synchronization process between the sub-chronicle as follows (see Figure 1, internal events are own events in sub-chronicle).

Definition 3 Variables State: shows if the value of a variable of an event in the chronicle is normal (¬Err) or abnormal (Err). Usually, we can denote it using a Boolean value (0 and 1), but the abnormal state may expand to enrich the classification of this behavior.

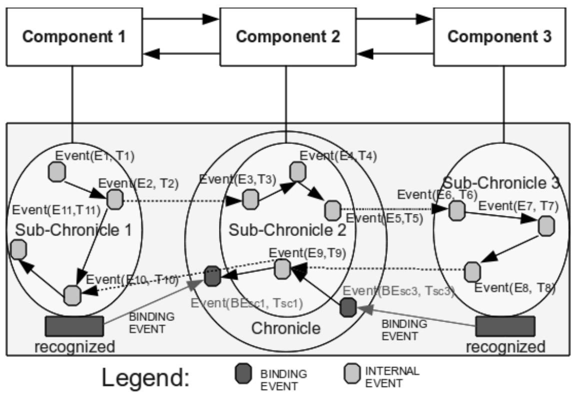

Fig. 1. Example of chronicle decomposed sub-chronicles

Definition 4 Binding Events (BE): Are events from sub-chronicles, to connect them to other sub-chronicles, and thus to represent the communication between sub-chronicles. A binding event BE_j is instanced when a neighbor sub-chronicle is recognized and then is propagated to the other sub-chronicles. Thus, the recognition (output event) of a sub-chronicle SC_i can be linked to the BE_j events belonging to a sub-chronicle SC_j

Definition 5: we define a Distributed Chronicle as "a chronicle C decomposed into a set of sub-chronicles SC_i, which are linked together via Binding Events"

Proof 5: To prove this claim, suppose the chronicle in Figure 1, this chronicle is decomposed in 3 sub-chronicles:

$SC_1 = \{(E_1, T_1), (E_2, T_2), (E_{10}, T_{10}), (E_{11}, T_{11})\}$

$SC_2 = \{(E_3, T_3), (E_4, T_4), (BE_{sc3}, T_{sc3}), (E_5, T_5), (BE_{sc1}, T_{sc1}), (E_9, T_9)\}$

$SC_3 = \{(E_6, T_6), (E_7, T_7), (E_8, T_8)\}$

SC_1 and SC_3 are sub-chronicles with only internal events. When SC_1 and SC_3 are recognized then they emit binding events (BE_{sc3} in time T_{sc3}, and BE_{sc1} in time T_{sc1}) to SC_2. Finally, SC_2 can recognize the global chronicle:

$SC_2 = \{(E_3, T_3), (E_4, T_4), (BE_{sc3}, T_{sc3}), (E_5, T_5), (BE_{sc1}, T_{sc1}), (E_9, T_9)\}$

$SC_2 = \{(E_3, T_3), (E_4, T_4), \{(E_6, T_6), (E_7, T_7), (E_8, T_8)\}, (E_5, T_5), \{(E_1, T_1), (E_2, T_2), (E_{10}, T_{10}), (E_{11}, T_{11})\}, (E_9, T_9)\}$

$SC_2(Eac_2, Tac_2), UNION_{j=1, 3 \mid j \neq 2}(SC_j(Eac_j, Tac_j)) = C(E,T)$

4.2 Distributed Chronicles Recognition

To design a fully distributed chronicle recognition, a recognition system is placed in each component of the system, This local recognition system is in charge to recognize the sub-chronicle associated to the component (i.e. site). As mentioned above, each sub-chronicle is linked to other events through binding events, which will allow to the local recognition system to infer information from its neighboring components. Using this information, the recognition system can recognize local sub-chronicles, generate events to neighboring sites, and in general, spread the inferred. The inclusion of binding events allows each site to have a global vision of the entire system.

For the chronicle example of Figure 1, the architecture of each site is shown in Figure 2, and consists of a local chronicle recognition module (called CRS), which receives events from the local monitor, and events generated by other neighboring sites (BE). For example, we see in Figure 2 as the site 2, in a given time, receives events from the monitor 2, and those generated by the sub-chronicles at sites 3 (BE_3) and 1 (BE_1). Then Chronicle Model of the Figure 2 can be written as shown on Figure 3.

4.2.1 Local chronicle recognition module algorithm

The local CRS performs diagnosis; the algorithm to make this is shown below:

```
1. FOREACH event received DO
2.  diagnoser.addEvents(chronicle C, event)
3.  IF chronicle C is recognized THEN
        3.1    DO C.executeAction()
```

Algorithm 1: local CRS

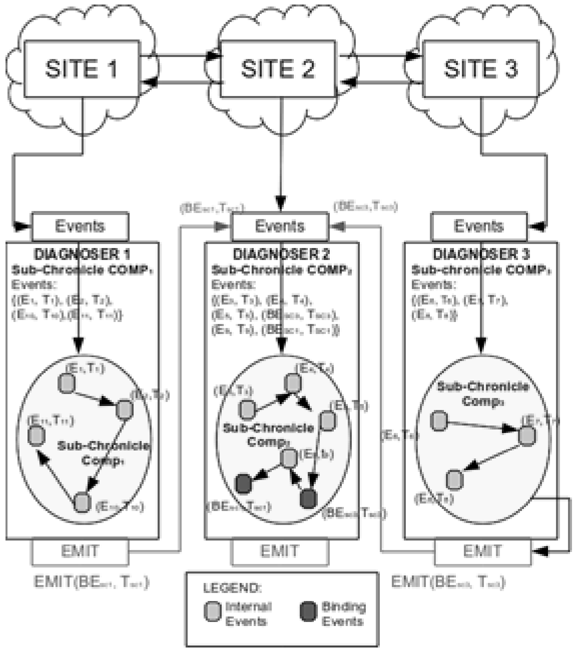

Fig. 2. distributed CRS

To illustrate the operation of the algorithm, we are going to describe every step using the chronicle model in the component 3 of the figure 3 using C9 = 1, C10 = 3, C11 = 1 and C12 = 3:

1 The events are represented by the name of the activity and time of event occurrence. Thus, a diagnoser will getting a stream of events that are induced by the evolution of the component or are generated by other diagnoser, to make the recognition of a chronicle. To illustrate this point let us suppose that diagnoser 3 receives the sequence of events:

{event(E6,T6=2), event(E6, T6=3), event(E7, T7=6), event(E7, T7=7), event(E8, E8=8)}

Chronicle Subchronicle 1 {
Events{
event(E_1, T_1), event(E_2, T_2), event(E_{10}, T_{10}), event(E_{11}, T_{11}) }
Constraints{
**T_2-$T_1 \leq C_1$
T_{10}-$T_2 \leq C_2$
T_{11}-$T_{10} \leq C_3$** }
When recognized{
Emit event(BE_{SC1},T_{SC1}) to Diagnoser 2 }}
	Chronicle Subchronicle 2 {
	Events{
	event(E_3, T_3), event(E_4, T_4), event(E_5, T_5), event(BE_{SC3}, T_{SC3}), event(E_9, T_9), event(BE_{SC1}, T_{SC1}) }
	Constraints{
	**T_4-$T_3 \leq C_4$
	T_5-$T_4 \leq C_5$
	T_{SC3}-$T_5 \leq C_6$
	T_{SC1}-$T_9 \leq C_8$** }
	When recognized{
	Create log(Fault 1)
	}}
		Chronicle Subchronicle 3 {
		Events{
		event(E_6, T_6),event(E_7, T_7), event(E_8, T_8) }
		Constraints{
		**T_7-$T_6 \geq C_9$
		T_7-$T_6 \leq C_{10}$
		T_8-$T_7 \geq C_{11}$
		T_8-$T_7 \leq C_{12}$** }
		When recognized{
		Emit event(E_{SC3},T_{SC3}) to Diagnoser 2 }}

Fig. 3. Chronicle Model

2 To illustrate the process of adding a new event to the instances of the chronicles, we detail this step in Algorithm 2:

```
diagnoser.addEvents (chronicle C, event)

1 FOR EACH chronicle C where event is the first event DO
1.1   instances = C.instances();
2   FOR EACH current instances DO
2.1  IF event matches instances and temporal constraints are
     not violated THEN
2.1.1   instances.addEvent(event)
2.2  IF event match instances and temporal constraints are
     violated THEN
2.2.1   instances.discard()
2.3  IF temporal constraints are violated THEN
2.3.1   instances.discard()
```

Algorithm 2: Procedure diagnoser.addEvents

At the time of arrival of an event, are created many instances of chronicles as different possibilities exist (this event is the first one of these chronicles, see step 2.1). Additionally, it is necessary to check all the current chronicles instanced (step 2.2) in order to determine if some of them can continue to be instanced (they wait for this event, see step 2.2.1) or in some of them the temporary constraints are violated (step 2.2.2). For this last case, these instances must be deleted. Of course, an instance could not be affected because the new event is not part of the chronicle model. Additionally, the chronicle instances can be discarded when no new events occur and the time constraints are violated (see step 2.3.1).

This behavior of the algorithm is shown in Figure 4 for the CRS of the diagnoser 3. Initially the set of hypotheses is empty i.e. there are no (partial) chronicle instance and arrives E6 event in the diagnoser at time T6 = 2; an instance I31 of the chronicle model is created waiting for the E7 event in the interval [3, 5]. Then arrives another event E6 at time T6 = 3 and a new instance is created I32 awaiting the event E7 in the range [4, 6]. At the time t = 5, I31 instance has a temporal constraint violation because the event E7 has not happen and then the I31 instance is discarded. At time t = 6 occurs the event E7 and Chronicle instance I32 is modified waiting the event E8 in the interval [7, 9]. E7 event arrives at time T7 = 7 but there are not instances did not occurred. Finally, E8 occurs and the instance of chronicle I32 is recognized.

3 The recognition of a chronicle is when for a stream of observable events, the full pattern of the chronicle model is reached. That is, for each instance of a chronicle in the CRS, this step must verify if the current event is the last event awaited. In this case, this instance becomes in a recognition of the chronicle. For the previous example, we observe in Figure 4 how subchronicle 3 is recognized in the I32 instance and it is not recognized in instance I31 (this instance is discarded):

I31 = {E6(T6=2)}

I32 = {E6(T6=3), E7(E7=6), E8(T8=8)}

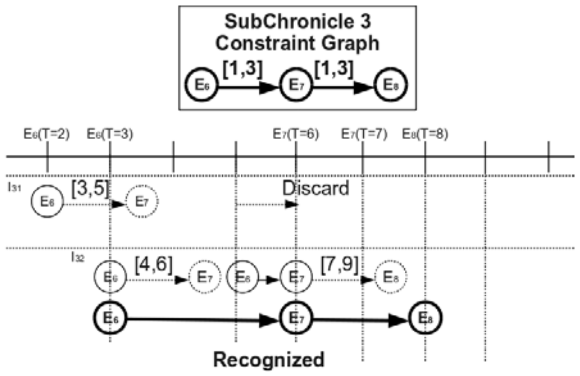

Fig. 4 Sub-chronicles 3 instanced

4 The recognition of chronicles is performed to identify undesirable behaviors in systems, so that the chronicle recognition induces the execution of a set of actions. These actions are not limited to the generation of data reports, but they may consist of corrective actions or generation of new events that could be used by other diagnosers. In our example, the chronicle model contains one action to be executed at the time of recognition:

Emit event(eSC3, Diagnoser 2)

That is, when SC3 chronicle is recognized one event eSC3 is generated and sent to Diagnoser 2.

5 Case Study

In order to illustrate our proposal, we will use a common example of e-commerce SOA application[1] (see Figure 5), which comprises three business processes (which will constitute our services):

- **Shop:** the shop where users purchase products.
- **Supplier** offers products to the store, it needs to check their availability before making a response to the store.
- **Warehouse:** where the products are stored in the providers. This process has a service level agreement (SLA)[2] with Supplier, which is that at least one product from the list should be returned[3]. It can invoke to other warehouse of the company to search products. This property allows to answer at least one product when the required amount is not in the local warehouse.

Now, we describe a classical behavior of this application:

(1) **SuppListOut:** Shop provides the list of products required to the supplier.
(2) **SuppItemIn:** Supplier checks its deposit invoking the Warehouse process.
(3) **SuppItemOut:** Warehouse provides the answer about the list of products in the deposit to the Supplier, which must contain at least one product (SLA restriction).

SuppListIn: The Supplier notifies the Shop which products can provide.

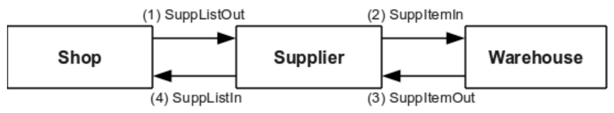

Fig. 5 e-commerce example

5.1 Design of Chronicles

Let us now characterize the distribution of events among different diagnosers (sites) that are part of the composition, With this generic chronicle we can derive each specific chronicle to detect abnormal situation. Additionally, for practical reasons we consider that the time is measu-

[1] SOA Application is a distributed application developed under SOA (Service Oriented Architecture).
[2] SLA is a contract between the service consumer and service provider and defines the level of service.
[3] This SLA defines how message delivery is guaranteed, the Warehouse delivery messages in the proper order (least one product in order).

red in seconds, and delay in communications and the recognition time of chronicles are negligible. The sequence of events in the generic chronicle is:

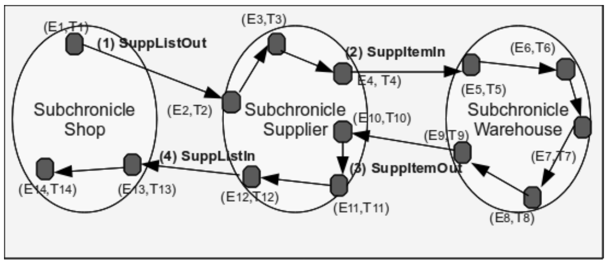

Fig. 6. Chronicle divided into sub-chronicles in the e-commerce example.

- **Shop Events:**
  (E_1) Shop sends product order to Supplier.
  (E_{13}) Shop receives the list of products.
  (E_{14}) Shop makes products payment.
- **Supplier Events:**
  (E_2) Supplier receives product order
  (E_3) Supplier checks the products in the catalog.
  (E_4) Supplier provides product order to Warehouse for the products that it has not.
  (E_{10}) Supplier receives the response of the products.
  (E_{11}) Supplier makes the invoice.
  (E_{12}) Supplier answers to shop with products shipped.
- **Warehouse Events:**
  (E_5) Warehouse receives the request of the Supplier.
  (E_6) Warehouse searches products (may be it invokes other warehouses).
  (E_7) Warehouse updates inventory.
  (E_8) Warehouse packs and ships products to the buyer.
  (E_9) Warehouse provides the answer about the list of products in the deposit to the Supplier

Now, we can define the specific chronicle for each faulty behaviour that we want to diagnose. We will consider the following failure scenarios:

- The failure because there is a violation of a Warehouse Service Agreement (SLA violation), it is a type of web service failure.
- The failure due to the time delay to provide a service (Warehouse Service Delay), it is a fault in the flow of choreography.

The detection of these two failures is interesting because they allow us to assess the ability of our system to detect faults, both of service (local) or of choreography (global).

From current events in the choreography, we build specific chronicles for the two faults. In general, each specified chronicle is decomposed into the same three sub-chronicles defined above. Sometimes, for a given situation maybe we can need less sub-chronicles, as in the case of the failures studied (see figure 7, which shows the structure of the specified chronicle for the Warehouse SLA Violation and the Warehouse Service Delay faults).

To design the Chronicle Model for SLA Violation Warehouse, we analyze events in the Warehouse: the problem of SLA violation occurs when the Warehouse Service performs the products search in the event E6 (Warehouse searches products) and it does not get anything, or it does not invoke another Warehouse (E7), or Warehouse fails in packs and ships some products to the buyer (E8)., and emits the response to Supplier with a list empty of products. Then, the Shop service detects a fault in the event E10 (Supplier receives the response of the products). For this reason, the sub-chronicle in Supplier sends the event BESC2Sea to Warehouse diagnoser (really, the CRS in supply diagnoser sends the message), and then this diagnoser can recognize the fault SLA Violation. When the Warehouse Diagnoser recognizes the chronicle, it invokes the repair with the fault found.

For the case of SLA Violation fault (case of products search), the chronicle would be the figure 8, where pl is the product list.

In the case of the second fault, the design of the chronicle "Warehouse: Service Delay" is much more complex (see figure 9). We consider that the service warehouse presents a delay when it emits the response to Supplier after 10 sec (it is allowed a delay of 10 sec) and this behavior is repeated more than 2 times in 150 sec. Thus, sub-chronicle supplier detects when E10 event does not occurs at time (before 10 sec) and sends the event ESC2SeaDelay to diagnoser Warehouse each time when it detects that. For Warehouse, it recognizes a fault when it receives more than 2 times the event BESC2SeaDelay and previously received event E5 (Warehouse receives the request of the Supplier). When Warehouse recognizes the sub-chronicle invokes the repair with the fault found.

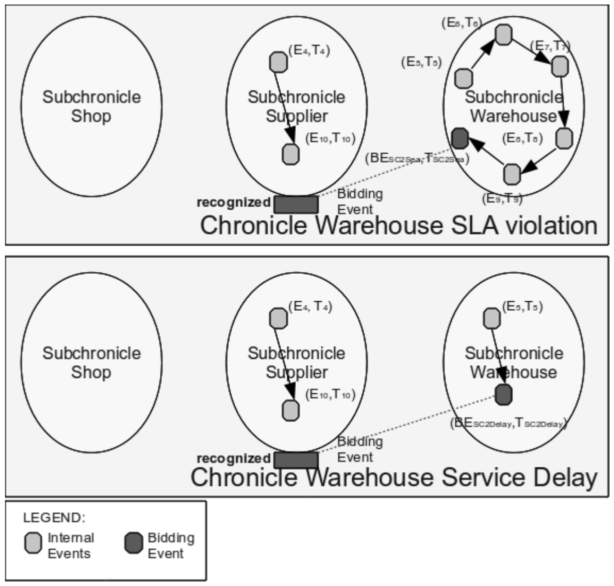

Fig. 7 Distribution of events in sub-chronicles on SLA Violation Warehouse and Warehouse Service Delay faults.

In the previous Chronicle, T10-T4 < 12 defines a maximal time to wait E10. Additionally, TSC2SeaDelay-T5 ≤ 13 is the time defined on the constraints on the Warehouse service for the arrival of their events plus one extra time defined by us as 6 (it is the time when the Warehouse Chronicle must receive BESC2SeaDelay event). Finally, * means any value. It is important to note that the Chronicle model when the warehouse does not respond is totally different, and is entirely managed by the Supplier Diagnoser.In the Chronicle of the figure 10, T4 + 12 due to the same reason to the previous Chronicle: the time defined on the constraints on the Warehouse service for the arrived of their events plus one extra time defined by us as 6.

Sub-chronicle Shop { Events{ } Constraints{ } When recognized{ } }	Sub-chronicle Supplier { Events{ **event(E_4: pl > 0, T_4)** **event(E_{10}: pl = 0, T_{10})** } Constraints{ **T_{10}-$T_4 \leq 9$** } When recognized{ Emit event(BE_{SC2Sea},T_{SC2Sea}, Diagnoser 3) } }	Sub-chronicle Warehouse { **Events{ event(E_5; pl > 0, T_5), event(E_6 : pl = 0, T_6), event(E_7: pl = 0, T_7), event(E_8: pl = 0, T_8), event(E_9: pl = 0, T_9), event(BE_{SC2Sea}, T_{SC2Sea})** } Constraints{ **T_6-$T_5 \leq 1$ T_7-$T_6 \leq 2$ T_8-$T_7 \leq 1$ T_9-$T_8 \leq 1$ T_{SC2Sea}-$T_9 \leq 1$** } When **recognized{ repairer Invoke(Warehouse, 'SLA-Violation'** }}

Fig. 8 Chronicle Model for SLA Violation Warehouse (case of products search)

Sub-chronicle Shop { Events{ } Constraints{ } When recognized{ } }	Sub-chronicle Supplier { Events{ **event(E_4; pl > 0, T_4), event(E_{10}; pl > 0, T_{10})** } Constraints{ **T_{10}-$T_4 \geq 10$ T_{10}-$T_4 < 12$** } When recognized{ **Emit event($BE_{SC2SeaDelay}$,$T_{SC2SeaDelay}$, Diagnoser 3)** } }	Sub-chronicle Warehouse { Events{ **occurs((3,100), {event(E_5 ; pl = *, T_5), event($BE_{SC2SeaDelay}$, $T_{SC2SeaDelay}$)}, (T_5, T_5+150))** } Constraints{ **$T_{SC2SeaDelay}$-$T_5 \leq 13$** } When **recognized{ repairer Invoke(Warehouse, 'Delay'** } }

Fig. 9 Chronicle Model for Warehouse: Service Delay

Events for the Shop Service: E_1(T_1 = 1, lp=3)
Events for the Supplier Service: E_2(T_2 = 2, lp=3), E_3(T_3 = 3, lp=3), E_4(T_4 = 5, lp=3), E_{10} (T_{10} = 12, lp=0)
Events for the Warehouse Service: E_5(T_5 = 6, lp= 3), E_6(T_6 = 7, lp=0), E_7(T_7 = 9, lp=0) , E_8(T_8 = 10, lp=0) , E_9(T_9 = 11, lp=0), E_{SC2Sea}(T_{SC2Ses} = 12)

For this events flow, the algorithm 1 recognizes a SLA Violation Warehouse fault on the Warehouse diagnoser using the chronicle for SLA Violation. The sequence of events that are detected and the recognition of instances of the chronicle are shown in Figure 11. The CRS Supplier Diagnoser recognizes the sub-chronicle "Warehouse SLA violation error" because is instanced {E_4(T_4 = 5, pl=3), E_{10}(T_{10}=12, pl=0)}, as Warehouse provides an empty list of products. Then, Supplier diagnoser emits the event ESC2Sea to Warehouse diagnoser, which recognizes the sub-chronicle instance {E_5(T_5 = 6, lp= 3), E_6(T_6 = 7, lp=0), E_7(T_7 = 9, lp=0) , E_8(T_8 = 10, lp=0) , E_9(T_9 = 11, lp=0), E_SC2S_{ea}(T_SC2S_{es} = 12)}.

The solution is to adjust the Warehouse service configuration to perform an external search of products in order to provide at least one product. Warehouse Repairer is invoked to perform the repair action (adjust external property) to ensure the proper functioning of the choreography.

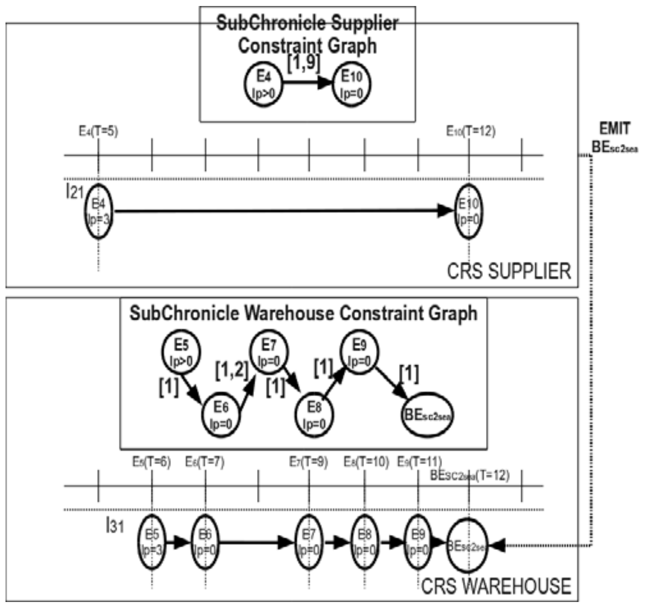

Fig. 11 Chronicles instances in Warehouse SLA Violation example.

5.3 Description of the chronicle recognition to detect Warehouse Service Delay

Let us consider now the following sequence of events in the e-commerce application (see figure 12) :

Events for the Shop Service: E_1(T_1 = 1, lp=1), E_1(T_{11} = 15, lp=2), E_1(T_1 = 26, lp=4).
Events (for the Supplier Service: E_2(T_2 = 2, lp=1), E_3(T_3 = 3, lp=1), E_4(T_4 = 5, lp=1), E_{10}(T_{10} = 15, lp=1), E_2(T_2 = 16, lp=2),

$E_3(T_3 = 17, lp=2)$, $E_4(T_4 = 19, lp=2)$, $E_{10}(T_{10} = 29, lp=2)$, $E_2(T_2 = 32, lp=4)$, $E_3(T_3 = 33, lp=4)$, $E_4(T_4 = 34, lp=4)$, $E_{10}(T_{10} = 45, lp=4)$.

Events for the Warehouse Service: $E_5(T_5 = 6, lp= 1)$, $E_6(T_6 = 9, lp=1)$, $E_7(T_7 = 11, lp= 1)$, $E_8(T_8 = 13, lp=1)$, $E_9(T_9 = 14, lp=1)$, $E_{SC2Delay}(T_{SC2Delay} = 15)$, $E_5(T_5 = 20, lp= 2)$, $E_6(T_6 = 21, lp=2)$, $E_7(T_7 = 26, lp= 1)$, $E_8(T_8 = 27, lp=2)$, $E_9(T_9 = 28, lp=2)$, $E_{SC2Delay}(T_{SC2Delay} = 29)$, $E_5(T_5 = 34, lp= 4)$, $E_6(T_6 = 40, lp=4)$, $E_7(T_7 = 41, lp= 4)$, $E_8(T_8 = 43, lp=4)$, $E_9(T_9 = 44, lp=4)$, $E_{SC2Delay}(T_{SC2Delay} = 45)$.

Supplier Diagnoser recognizes 3 instances; I_{21}: {E4(T4 = 5), E10, (T10 = 15)}, I22: {E4(T4=19), E10, (T10 = 29)} and I23: {E4(T4 = 34), E10, (T10 = 45)}. Then, Supplier diagnoser emits the event $EB_{SC2SeaDelay}$ 3 times to Warehouse diagnoser to tell it that there is a problem of response delay. In this way, Warehouse diagnoser can recognize the sub-chronicle due to the next sequence of events {E5(T5 = 6), $E_{SC2Delay}(T_{SC2Delay} = 15)$, $E_5(T_5 = 20)$,,$E_{SC2Delay}(T_{SC2Delay} = 29)$, E5(T5 = 35), $E_{SC2Delay}(T_{SC2Delay} = 45)$}, because it occurs more than 2 times in less than 150 sec. Then, the Warehouse Diagnoser can invoke its repairer to perform the repair action of substitution of the warehouse service.

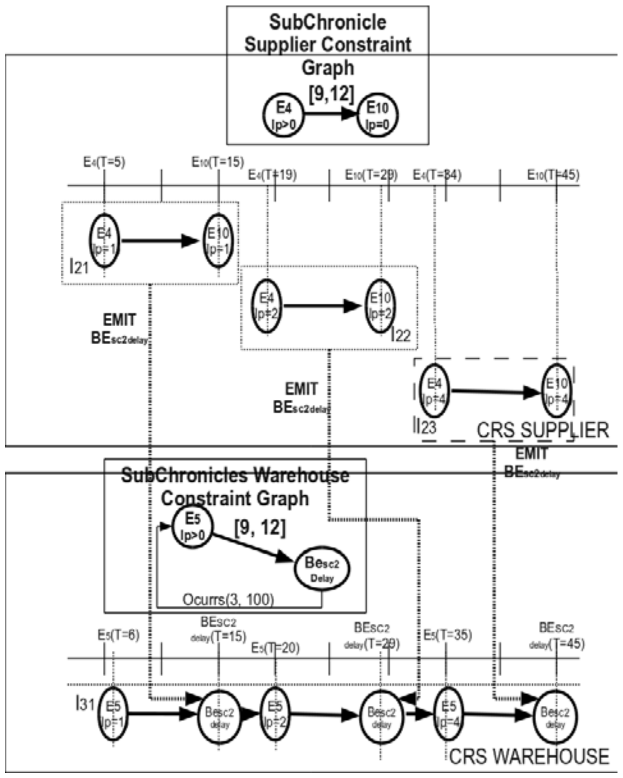

Fig. 12 Chronicles instances in Warehouse Delay example.

5.4 Analysis of results

Our chronicles detect the faults in the system. In the first case, at the level of a service failure (problems in it, which results in a local failure); and in the second case, a failure in the choreography (characteristic of distributed environments). This shows the versatility of our approach to address these different aspects in a distributed application.

The extension of the formalism of chronicles and the distribution of the CRS (Chronicle Recognition System) between the services that are part of the choreography, facilitate the interaction of the local Diagnosers, making the recognition of the global chronicle without need a coordinator to manage their interactions. At communication level, this represents a remarkable improvement over the mechanisms shown in other studies (WS-Diamond 2008, Cordier y col., 2000, Cordier y col., 2007, Quiniou y col., 2001, Dousson 2002). Additionally, the implementation of the mechanism in the cases studied is natural (a recognizer by service). Also, being a distributed approach, the scalability problem of the distributed application can be handled properly by our approach.

We compare our recognition mechanism for our fully distributed chronicles approach with the propositions in (Le Guillou y col., 2008, Boufaied y col., 2004), to calculate the amount of events exchanged between the different diagnosers and the amount of processed events for the global recognition of a chronicle. For this, we assume that this global chronicle is composed of m events distributed between n diagnosers (each diagnoser has in average m/n events which are necessary for the recognition of its chronicles), where m ≥ n. Table 1 shows the calculation of the metric to compare different architectures.Table 1 shows that our proposal requires only n-1 events to exchange for the global diagnosis of chronicle, against the other architectures that require more events. Moreover, our architecture needs to process fewer events for the global diagnosis (m+n-1 events), against the events processed by the other architectures. The reason our recognition mechanism requires less events to exchange and process less events, is because in our architecture the calculation is fully distributed, minimizing the number of events required for global diagnosis, and therefore the number of events to process, making it more scalable in its implementation and ideal for diagnosing distributed system with large number of component systems.

Table 1 Compare our architectures.

Architecture	events exchanged	Processed Events (Global constraint + Local constraint)
Decentralized Architecture (Le Guillou y col., 2008)	$n*\left(\frac{m}{n}\right)=n$	$\left(n*\left(\frac{m}{n}\right)\right)+(m)=2m$
Distributed Architecture "Event Spread" (Boufaied y col., 2004)	$(n-1)*\left(\frac{m}{n}\right)$	$(m)+\left((n-1)*\left(\frac{m}{n}\right)\right)=\frac{(m*(2n-1))}{n}$
Our Proposal	$n-1$	$\left(\left(\frac{m}{n}\right)+(n-1)\right)+\left((n-1)*\left(\frac{m}{n}\right)\right)=m+n-1$

6 Conclusion

In this paper we propose a distributed mechanism based on chronicles, which allows the distribution of the recognition of the possible faults in SOA applications, which favors its implementation in large systems. We have extended the formalism of chronicles, introducing the notion of sub-chronicles, binding events, etc. Furthermore, we have described the process of recognition of our model of chronicle fully distributed. Our distributed approach contrasts with the semi-centralized and decentralized ap-proaches that have been developed so far. In the case study, we test the distributed recognition mechanism to detect failures in two situations, at the level of the service (locally), and in the composition (globally). Additionally, we have shown in the case study that our recognition mechanism is simple to implement. Particularly, the fact to operate fully distributed is a notable advance in this area.

Thus, the recognition mechanism of distributed chronicle proposed in this work can be used not only in the management of faults in web service composition, but in other areas that require the distributed temporal pattern recognition, such as distributed simulations, community of software agents, etc.

Futures works need test our approach in real buses of services like OpenESB, this requires the implementation of the Chronicles using the Intelligent Event Process (IEP) component of OpenESB. IEP allows to event management using SQL (Structured Query Language). Thus, for each sub-chronicle is necessary to implement an IEP for events recognition and a communication mechanism that allows the distribution of Bidding Event between the different sub-chronicles, in order to reach a global recognition.

7 Acknowledgment

This work has been supported by FP7-ICT IMAGINE research project (European Commission, Grand Agreement No: 285132), and the PCP program “Supervision and maintenance tasks in a shared organizational environment”. Dr Aguilar has been partially supported by the Prometeo Project of the Ministry of Higher Education, Science, Technology and Innovation of the Republic of Ecuador.

References

Aghasaryan A, Fabre E, Benveniste A, Boubour R, Jard C, 1998, Fault detection and diagnosis in distributed systems : an approach by partially stochastic petri nets, Discrete Event Dynamic Systems, Vol. 8, No. 2, pp. 203-231.

Boel, R, van Schuppen, J, 2002. Decentralized failure diagnosis for discrete-event systems with costly communication between diagnosers. Proc. Sixth IEEE International Workshop on Discrete Event Systems, pp. 175-181.

Boufaied A, Subias A, Combaceau M, 2004, Distributed fault detection with delays consideration, Proc. of the 15th Int. Workshop on Principles of Diagnosis.

Cordier, M, Dousson C, 2000, Alarm driven monitoring based on chronicles, Proc. of Safeprocess’2000, pp. 286-291.

Cordier M, Le Guillou X, Robin S, Roze L, Vidal T, 2007, Distributed chronicles for on-line diagnosis of web services. Proc. of 18th International Workshop on Principles of Diagnosis, pp 37–44

Dousson C, 2002, Extending and unifying chronicle representation with event counters, ECAI, pp. 257-261.

Dousson C, Gaborit P, Ghallab M, 1993, Situation recognition: representation and algorithms, Proc. of the Int. Joint Conf. on Artificial Intelligence, pp. 166-172.

Guerraz B, Dousson C, 2004, Chronicles construction starting from the fault model of the system to diagnose, Proc. of the 15th Int. Workshop on Principles of Diagnosis, pp. 51-56.

Grosclaude I, 2004, Model-based monitoring of component-based software systems, Proc. of the 15th Int. Workshop on Principles of Diagnosis, pp. 51-56.

Le Guillou X, Cordier MO, Robin S, Rozé L, 2008, Chronicles for On-line Diagnosis of Distributed Systems, Proceeding of the 18th European Conference on Artificial Intelligence, pp.194-198.

Le Mortellec, A, Clarhaut, J, Sallez, Y, Berger, T, & Trentesaux, D, 2013, Embedded holonic fault diagnosis of complex transportation systems. Engineering Applications of Artificial Intelligence, Vol. 26, No. 1, pp. 227-240.

Mhalla A, Jerbi N, Collart S, Craye E, Benrejeb M, 2010, Distributed Monitoring Based on Chronicles Recognition for Milk Manufacturing Unit, Journal. of Aut. & Syst. Eng, Vol. 4 No. 1.

Morin B, Debar H, 2003, Correlation on intrusion: an application of chronicles, 6th International Conference on recent Advances in Intrusion Detection RAID, Pittsburgh, USA.

Nakata, S, Takai, S, 2013, Reliable decentralized failure diagnosis of discrete event systems. SICE Journal of Control, Measurement, and System Integration, 6(5), pp. 353-359.

Quiniou R, Cordier M, Carrault G, Wang F, 2001, Application of ilp to cardiac arrhythmia characterization for chronicle recognition, ILP’2001, pp. 220–227.

Roychoudhury, I, Biswas, G, & Koutsoukos, X., 2009, Designing distributed diagnosers for complex continuous systems. IEEE Transactions on Automation Science and Engineering, 6(2), pp. 277-290.

Vizcarrondo J, Aguilar J, Exposito E, Subias A, 2012, ARMISCOM: Autonomic Reflective MIddleware for management Service COMposition, Proceedings of the 4th Global Information Infrastructure and Networking Symposium (GIIS 2012), IEEE Communication Society, Choroni, Venezuela.

WS-Diamond, 2008, WS-Diamond, IST-516933, Deliverable D4.3, Specification of diagnosis algorithms for Web Services – phase 2. Version 0.5.

***Recibido:** 15 de julio de 2014*

***Aceptado: 26** de marzo de 2015*

***Vizcarrondo, Juan**: is System Engineer and obtained a Msc in Computer Science at the Universidad de los Andes, Mérida-Venezuela, Currently he's finished his studies PhD in Computer Science at the Universidad de los Andes. He works at the Cenditel since 2007.*

***Aguilar José:** is System Engineer from the Universidad of the Andes-Mérida-Venezuela, obtained a Msc in Computer Science at the University Paul Sabatier-Toulouse-France, and a PhD in Computer Science at the University Rene Descartes-Paris-France. Also did a Postdoctoral in Department of Computer Science at the University of Houston. **Email:** aguilar@ula.ve*

***Exposito, Ernesto:** earned his engineer degree in computer science from the "Universidad Centro-occidental Lisandro Alvarado" (Venezuela, 1994). He earned his PhD in "Informatique et Télécommunications" from the Institut National Polytechnique de Toulouse (France, 2003). Email: ernesto.exposito@laas.fr*

***Subias Audine**: received a PhD degree in 1995 and a M.S. degree in 1992 in Informatique Industrielle, both from Paul Sabatier University, in Toulouse, France. Since 1997 she is Associate Professor in control and discrete event systems at the Institut National des Sciences Appliquées (INSA) of Toulouse**.** Email: subias@laas.fr*