DOTTORATO DI RICERCA

IN

SCIENZE COMPUTAZIONALI E INFORMATICHE

CICLO XIX

CONSORZIO TRA UNIVERSITÀ DI CATANIA, UNIVERSITÀ DI NAPOLI FEDERICO II, SECONDA UNIVERSITÀ DI NAPOLI, UNIVERSITÀ DI PALERMO, UNIVERSITÀ DI SALERNO

SEDE AMMINISTRATIVA: UNIVERSITÀ DI NAPOLI FEDERICO II

ROSA PALMIERO

# A Subscription Language For Event-Based Monitoring of Grid Resources and Services

*Tesi di Dottorato di Ricerca*

Il Coordinatore

Prof. Aldo de Luca

# Contents

# Acknowledgments

I would like to express my gratitude to everyone who contributed to my research work: researchers and friends. In particular I would like to thanks Natascia De Bortoli for her suggestions to face this research work, Leonardo Merola, for his scientific and human support and Gennaro Tortone that actively collaborated to the work done.

*to Maria, my Mother*

# Foreword

At present the WEB allows the publication of data concerning everything. This is the time of *information data* moving online; much of information consists of data without any predefined structure which grows quickly causing a corresponding increase of complexity and size of the Web sites; much of it is expressed using XML metalanguage. Documents published rather than being manually composed, are usually automatically generated from the system. So, the documents hide a common structure that should emerge. In this context three different philosophies seem to enter in collision: the philosophy that *everything is a document*, the philosophy that *everything is an object* and *everything is a relation*. Each attitude produces a different culture, but each of them is able to explain an aspect of a such complex scenario. In this work, I approach a real problem: how to explain generic events produced by a system and how to query them. The produced events are a concrete example of unstructured data, to be converted in semistructured data; queries to be done follow the same requirement. The considered problem is common to many contexts as explained in the introduction. We try to give an answer to this question, building and describing a possible formalization.

# Preface

## 0.1 The context

This work was born in the frame of *Grid Monitoring*. It regards specifically design and development of a monitoring system for GRID technology, developed in part at National Institute of Nuclear Physics (INFN) of Naples. The goal consists in the formalization of interests of GRID users in the occurrence of some specific events, developing a language that can help them to submit their demands. In this thesis, formal definition of events and of subscriptions are given; in particular temporal subscriptions, interested in occurrence of events dependent on time, are formalized.

## 0.2 The aim

Even if the research work was born in a specific environment, it can be formalized independently from the problem that generated it. In this way, the obtained formalization can describe every context where users are interested in publication of some events and in the variation of some contained attributes. Let us suppose to manage a net bookstore, where events regarding variation of one or more attributes are published, such as, for example, the variation of books price, the entry of a new book's edition, new entries in the

bookstore and so on. For example, let us consider that a user is interested in the reduction of the price of a specific book within a specific temporal interval, beginning from the instant of subscription emission. This example, changing environment and attributes meaning, can describe various contexts such as Economy or Physics.
This thesis consists of two parts. In the first part, monitoring and management activities, applied to distributed systems, are introduced, then the generic event and the generic subscription, depending on the time, are formalized. In this part atomic, primitive and composite events, simple and complex subscriptions are structured; the functions that take events set as input and give contained attributes are defined. In the second part, the adopted formalization is applied to the Grid monitoring context. Language, which has been chosen to represent occurred events, is the Extensible Markup Language (XML); a query language chosen to explain subscriptions is the Extensible Path Language (XPath). In the specific case our goal is to explain queries, depending on the time, in an XPath extension. The choice of these languages does not cause loss of generality of the formalization realized.

### 0.2.1 Thesis structure

This thesis, as previously said, is divided in two parts.

The first part is divided in five chapters:

- Chapter I, Monitoring in Distributed Systems;
- Chapter II, The Content-Based Publish/Subscribe Paradigm
- Chapter III, Primitive Events: Formalization and Functions Definition;

- Chapter IV, Ennary Trees;
- Chapter V, Composite Events and Related Functions

The Second part is divided in three chapters

- Chapter VI, Semi-structured data and the extensible mark-up Language
- Chapter VII, the Pseudo-code: an idea of algorithm
- Chapter VIII, The recent emergent technology: Computational Grid

# Chapter 1

# Monitoring in Distributed Systems

## 1.1 Overview

Monitoring is the activity of obtaining, collecting, and presenting the information required by an observer about the observed system [Bib.1], [Bib.2], [Bib.3]. In this scenario the goal consists of obtaining information in order to describe a model of system behaviour or to modify an existing model (management role) [Bib.5]. Monitoring is essential to obtain the required information about the operations of distributed systems in order to make management decisions and to control their behaviour [Bib.4]. The manager (human or machine), on the base of monitoring information, modifies the state of the monitored system (management activity) [Bib.4]. Monitoring activity is necessary for various purposes. The general activity can be specialized as follows [Bib.5], [Bib.6]:

- debugging

- testing
- accounting
- performance evaluation
- resource using analysis
- security
- fault detection
- teaching aid

In this thesis we are interested in monitoring distributed systems. The characteristics of distributed systems, such as heterogeneity, autonomy, physical separation and concurrency, make the process of monitoring very complicated. So each design and development of monitoring facilities needs to deal with these problems. Management structures are, therefore, necessary to maintain information on the objects participating in a monitoring session, and manage the monitoring facilities in each of them. Monitoring information can be ordered in four phases [Bib.7]:

- generation
- processing
- dissemination
- presentation

A typical distributed system consists of a number of processes which run on different machines and cooperate to perform a common task. Processes

coordinate their activities by sending messages over a communication network. Examples of such systems include process control, telecommunication and banking applications.

## 1.2 Monitoring and Management activity

Monitoring activity is required for various purposes such as debugging, testing and program visualization. It is also used for general management activities such as: i) performance and quality of service management; ii) configuration management; iii) fault management; iv) security management; v) accounting management. A managed distributed system consists of a number of managed objects. We consider a managed object any hardware or software component whose behaviour can be monitored by a management system.

## 1.3 The problem of distributed systems

There are many fundamental problems associated with monitoring of distributed systems [Bib.50]. For example, the delays in transferring information from generator to the disseminator. This means it is very difficult to obtain a global and consistent view of all components in a distributed system. In order to overcome these problems, it is necessary to design a monitoring system in terms of a set of general functions related to generation, processing, distribution and presentation of monitoring information.

## 1.4 Filtering and Correlation Capabilities

In large distributed systems, the amount and frequency of the monitoring information quickly grow and cause managers to be overloaded with work. Sometimes a significant portion of this information may be irrelevant. An efficient monitoring service must realize filtering and correlation of reports with the aim of raising the abstraction level of monitoring information. In this thesis the terms correlation, combination and composition are used interchangeably.

## 1.5 Monitoring Models

According to Event Management Model [Bib.5], it is possible to identify the following four monitoring activities performed in a loosely-coupled, object-based distributed system:

- i) Generation: important events are detected and status reports are generated. These monitoring reports are used to construct monitoring traces, which represent historical views of system activity;
- ii) Processing: a generalized monitoring service provides common processing functionality such as merging of traces, validation, database updating, combination/correlation and filtering of monitoring information. They convert the raw and low-level monitoring data to the required format and level of detail;
- iii) Distribution: monitoring reports are distributed to users, managers (or processing agents) who require them;
- iv)Presentation: gathered and processed information are displayed to

the users in an appropriate form;

The most frequently model for analyzing distributed system behaviour consists of event-driven monitoring [Bib.8], [Bib.9]. In this model, the system behaviour is described using primitive events. Generic users subscribe their interest in the occurrence of composite or atomic events (composition is realized trough logic operators). If subscribed events occur, users are notified. In this context, time is fundamental. A composite subscription is satisfied if events pattern occurs. If only one pattern is registered in delay, the occurred events cannot be notified with the related error in the following actions. For example, if a subscriber is interested in occurrence of event: *e1*, $n$ seconds after of the occurrence of *e2*, where:

$$n \geq 0$$

a delay in registration of the event or a change in its occurrence, produces an error that could be fatal for the management activity. So, the definition of global time and the definition of good methods describing temporal restrictions in distributed environment, are essential in the monitoring phase. Distributed systems demand a dynamical monitoring activity because of their inner nature. The dynamic aspect is essential in simple operations such as: connection of one or more components, submission of new subscriptions or deleting of the old ones .

## 1.6 Formal environment description

In this section, we give the characteristics of the environment where a distributed system is monitored on the base of its status and occurred events. Let us define the system as "a group of independent but interrelated elements"

[Bib.10], or a collection of hardware and software components organized to accomplish a specific function or a set of functions [Bib.11]. Let us suppose this system is under observation through use of persistent sensors, able to gather all information useful for monitoring and management activities. The described environment consists of the system, all sensors and a set of rules to make management decisions, which may modify system's behaviour. More in details, gathered data are reported in the *Monitoring Reports*, used by automated or human manager to control system. Each component object is characterized by a status and a set of events (representing status changes and activities). The object behaviour can be observed and described in terms of both its status and events. The object status is a measure of its behaviour in a specific instant time, it can be represented using a vector of status variables (or attributes). These attributes may be static (for example, operative system installed) or time-varying (for example, CPU load). An event is defined as an entity that registers a modification of an object status. The status of an object is characterized by life time, for example, "process is idle", whereas an event occurs instantaneously. The status is usually continually changing so the object behaviour is monitored on the base of some particular events, called events of interest. It is necessary to distinguish between time-driven monitoring and event-driven monitoring. Time-driven monitoring is based on acquiring periodic status information to provide an instantaneous view of the behaviour of an object or a group of objects, while and event-driven monitoring is based on obtaining information about occurrence of events of interest, which provide a dynamic view of system activity. In the first approach, there is a direct relationship between the sampling rate and the amount of information generated. In the second one, the amount of generated and communicated monitoring data is reduced as only the information per-

taining to activity of interest. For this reason, the event–driven monitoring is the most common approach adopted in monitoring systems.

# Chapter 2

# The Content-Based Publish/Subscribe Paradigm

## 2.1 Overview

In this chapter it is necessary to introduce the publish/subscribe communication paradigm, because of the important role it plays in the frame of distributed systems and especially in this thesis. In the following sections we describe the fundamental characteristics of a distributed *Publish/Subscribe model*, we discuss the *content-based variant* and justify both the choices.

## 2.2 Introduction to a Publish/Subscribe Model

Nowadays the distributed systems are composed by thousand of components generally located all over the world. The location and the behaviour of each component determine the behaviour and even the lifetime of distributed system. For this reason the designing and developing of a more flexible communication model appear absolutely necessary, especially to preserve the

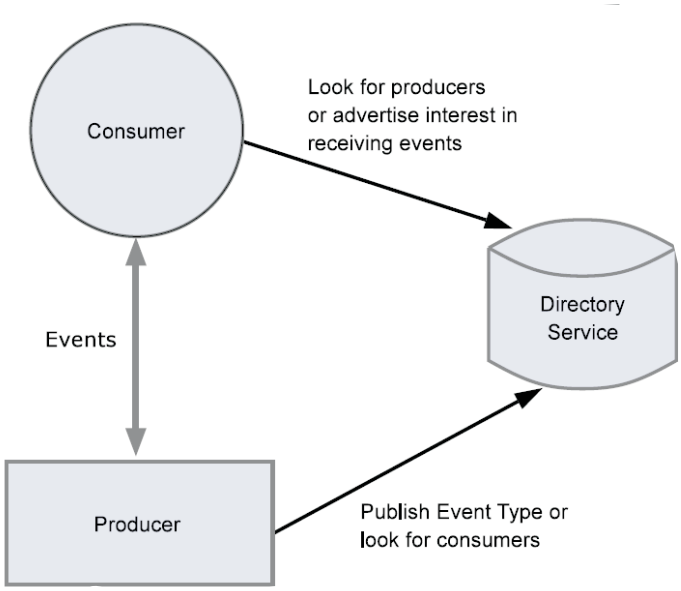


Figure 2.1: The producer and the consumer of event

dynamic and decoupled nature of involved applications, necessary for distributed systems [Bib.12]. In this scenario the publish/subscribe (P/S) schema seems to answer the many requirements previously advanced, for example, it provides a loosely coupled interaction. This communication paradigm provides two principal players, the publisher (the information producer) and the subscriber (the information consumer) (fig.1). The publisher entity publishes the information produced (the events) on a software bus, while the subscriber expresses her/his interest in a specific part of information (an event or in a pattern of events), submitting related subscription. The subscriber is notified when the submitted subscription matches an occurred event or a pattern of events. This communication model provides a simple and efficient methods for distributing information, moreover it guarantees decoupling of involved players. In fact in the P/S model the event is asynchronously sent to every subscribers interested in it, moreover a full decoupling in terms of time, space

and synchronization, between publisher and subscriber, is guaranteed.
In our research work we assume the subscriber can only subscribe to future data [Bib.12]. She/he can not receives information published before her/his subscription is submitted.

## 2.3 Description of interaction scheme and comparison with traditional ones

As we said, this interaction model provides a producer, that publishes the occurred events on a software bus, and a consumer, that expresses her/his interest in an event or a pattern submitting related subscription to the same bus. As we said, the action of sending the event to the interested subscriber is called notification.

The communication model is governed by a notification service that stores and manages submitted subscriptions and, in case of matching, provides to notify interested subscribers [Bib.13]. The presence of the event service

produces the decoupling between publisher and subscriber (fig.2). In fact each player can interact with service independently from the other. The decoupling can be interpreted under three different point of view: space decoupling, time decoupling and synchronization decoupling [Bib.12]. Let us analyze the meaning of a such decomposition:

- space decoupling: publisher and subscriber do not know each other, the publication and subscription are transparent and independent.
- time decoupling: the actions of publisher and subscriber can be done at different moments, they have not to interact, at same time, with event service

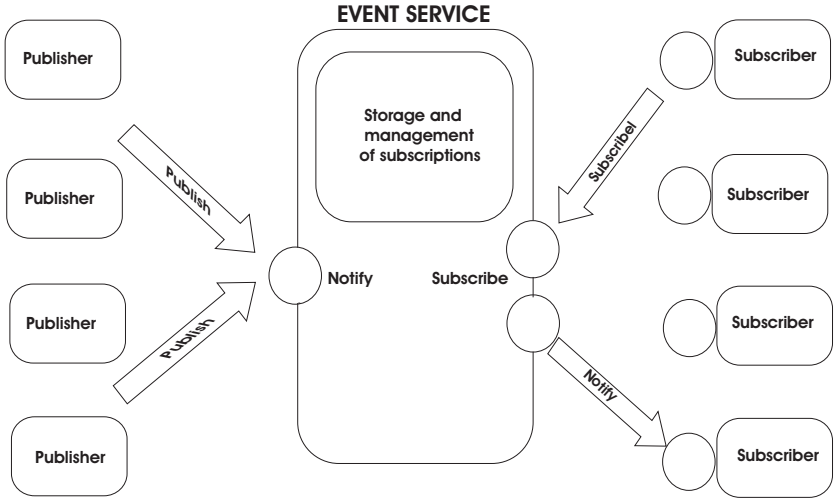


Figure 2.2: The The event service

- synchronization decoupling: the publishing and notifying operations are respectively asynchronous. If the producer is performing some activities and an event occurs, it can finish its activities and then publish the event, In the same way, the subscriber, that is engaged in other operations, can be notified when she/he has finished.

The alternative communication paradigms as *message passing*, *remote invocation*, *notifications*, *message queuing* and *shared space*, even though presenting same common points with P/S model, are all unable to guarantee a full decoupling between interacting parts. Traditional interaction models are not able to contemporary offer time, space and synchronization decoupling. This circumstance limits their implementation in distributed environment.

## 2.4 Description of Content-Based Model

In all distributed environments, subscribers are not interested in the set of all published events, usually their interests fall on a small subset [Bib.15].

As we said, they declare their interests through subscriptions, so the way of characterizing the waited event, defines the adopted subscription scheme. The most widely used schemes are the topic based and the content based [Bib.12].

In the topic based several topics (groups) are identified. The consumer information subscribes to one or more topics, becoming a member of that group (groups). The published events are distributed for topics and consequently broadcast to each group's member. The topic-based publish/subscribe model, is demonstrated to be static and primitive but simple to implement.

The content-based scheme is based on the content of the event [Bib.16]. The event is classified according to its properties, internal and external. Subscribers express their interest by a subscription language that allows to filter published information. For example, the subscription contains name-value pairs of properties composed with comparison operators (=, >, <, <=, >=) [Bib.12], [Bib.52]. The final content-based P/S paradigm, obtained adopting the content-based subscription scheme, is much more expressive of topic one, but, requiring sophisticated protocols [Bib.51], it is not so simple to implement. Even so, the content-based P/S is the most adaptable paradigm to distributed systems because of the expressive power of its model and subscription language [Bib.14], [Bib.56]. For this reason but especially because of the many studies regarding the connection between Grid environment and the content-based scheme [Bib.13], [Bib.17], we develop our work adopting this communication paradigm.

In this thesis we are not interested in the routing modality for produced messages. Generally the form of routing adopted in the networks is unicast or multicast. Nevertheless, in the last years, the interest of developers community, for the addressing and routing communication, is grown, specially

in the case of loosely coupled distributed systems with large number of consumer, with different interests and heterogeneous resources. For this reason we choose as communication model the content-based addressing and routing paradigm [Bib.57], [Bib.18], [Bib.19], [Bib.17], [Bib.16].

Most of existent content-based P/S systems are developed on the base of subscription languages. In the following chapters, we describe our events and subscriptions formalization that allows subscribers to submit to the system very complicated subscriptions, containing also time requests.

# Chapter 3

# Primitive Events: Formalization and Functions Definition

## 3.1 Chapter's Content

In this Chapter we introduce the basic element of our work: the event. Once the definition of the event is given, we introduce the related functions, useful to extract event attribute values, and the subscription definition. In the end of this chapter, some words will be spent to clarify the meaning that time assumes in this context.

## 3.2 Primitive and composite events

The events, the kernel of this research work, are here distinguished in primitive and composite. We assume that a primitive event occurs when a characteristic of a monitored object changes. A composite event is defined as

combination, or composition of primitive events [Bib.6]. Furthermore, composition of composite events gives composite events. All events are to be intended independent from each other and instantaneous. Each event happens in the same instant when a sensor registers a modification of one of the related attribute characteristics. The publication of the events happens each $\Delta T$ temporal interval. In the formalization developed an event (composite or primitive) is represented by a set of n–tuples as specified in following.

## 3.3 Primitive event definition

Let's define a primitive event as a set composed by a finite n–tuple of elements such as *name*, *time stamps*(tms) and a certain number of ordered triples containing *type, attribute, value* [Bib.16]:

$$event := \{(name\text{-}event, id\text{-}number, tms, (type, attribute, value), ..., (type, attribute, value))\}$$

with the following meaning:

- name-event:= name of the event
- id-number:= a unique number that identifies the occurred event
- tms:= when the event service publishes the event

while ordered triple (type,attribute,value) has the following meaning:

- type:= type of event, chosen between standard type
- attribute:= attribute name of the event
- value:= attribute value

In the previous definition, type falls in a set of primitive types commonly used in programming languages or query languages. The given definition is commonly used in technical literature, [Bib.16]. It is important to observe that temporal instant *tms* and each temporal instant we are going to use, are to be understood as an interval: [tms-h, tms+h], where $h$ is a defined value. Let us define a simple event or atomic event as an event containing only one triple:

$$event := (name\text{-}event, id\text{-}number, tms, (type, attribute, value), ..., (type, attribute, value))$$

Example: atomic event

$$(thermodynamic, id\text{-}number, 12:31, (float, thermodynamic, "2.0"))$$

If the user is interested in the occurrence of one or more events, she/he will submit to the monitoring system one or more subscriptions. A subscription declares the interest of the subscriber user in the occurrence of certain events. Through the use of binary operators, a subscription specifies attributes and restrictions on the assumed values. I am going to define subscriptions in the next sections. Here, for example, a simple request is reported:

$$(thermodynamic, id\text{-}number, t \geq Time{-}Of{-}Emission, (float, thermodynamic \mid x \geq 1.5))$$

Let us observe the Time-Of-Emission variable expresses the time of subscription submission. In the previous demand, the user is interested in the ocurrance of all events whose name is *thermodynamic* and whose value is greater then 1.5, published every time after the emission of subscription (tme). We observe that for many authors the terms event and notification are changeable. In this work I distinguish two terms, giving to them two different meanings as in the natural language: a notification is defined as the information sent to a user who submitted a subscription; the aim of notification is to advertise a user that the event of his/her interest occurred.

## 3.4 Subscription Definition

As we said in the previous chapter, the adopted paradigm is the publish/subscribe. In this model, the producer (publisher) submits her/his data (event) to the system while the information consumer (subscriber) indicates her/his interest in the occurrence of certain events. The set of indications provided by subscriber, written in the established format, constitutes the subscription. To represent a generic subscription I used the same structure adopted for the event, so a subscription is defined as a finite t-ple of elements as *name*, *time of emission*(tme) and a certain number of ordered quadruples containing *type, attribute, operator, value* [Bib.16]:

$$s := ((name_s, id\text{-}number_s, tme_s, (type_s, attribute_s, operators_s, value_s)))$$

where the elements contained have the following meaning:

- name:= name of the event of interest
- id-number:= the unique number to identifie the subscription and, finally, related subscriber
- tme:= time of the emission of subscription

while the ordered quadruple (type, attribute, operator, value) contains:

- type:= type of the event of interest
- attribute:= attribute name of the event of interest
- operators:= the comparison operator between values.
- value:= attribute value of interest

## 3.5 Relation between events and subscriptions

A primitive event is a set of t-ple containing a name, an id-number, a tms and a certain finite number of triples: (type, attribute, value). The event name is a string of characters. The attribute (or equally attribute name) contained in the triple is characterized by type and value. The reported types fall in a predefined set of types with a predefined set of operators. A simple subscription is represented as follows:

$$s := ((name_s, id\text{-}number_s, tme_s, (type_s, attribute_s, operators_s, value_s)))$$

a simple or atomic event is:

$$e := ((name_e, id\text{-}number_e, tms_e, (type_e, attribute_e, value_e)))$$

we say:

$$s \sqsubset e$$

(where the operators among the event and the subscription declares subscription is satisfied by event $e$) that $e$ satisfies the constraint contained in $s$, if:

$$name_s == name_e$$

$$tme_s \leq tms_e$$

$$type_s == type_e$$

$$attribute_s == attribute_e$$

$$operators(value_s, value_e) = True$$

Something about the covering operator and the connected theory is reported in appendix A.

## 3.6 Function defined on simple events

In this section, we define functions that are able to match published events and submitted subscriptions. Let us start by defining these functions on the set of all atomic events named *Eatomic*. Given an atomic event:

$$e := (name_e, id\text{-}number_e, tms_e, (type_e, attribute_e, value_e))$$

Let us define the function

$$Att : E_{atomic} \longrightarrow Attribute$$

where the co-domain is the set of all attributes, the function *Att* takes an event as input and gives the attribute contained in it

$$Att(e) = attribute_e$$

In the same way:

$$Val : E_{atomic} \longrightarrow Value$$

*Val* takes an atomic event as input and gives its value as output.

$$Val(e) = value_e$$

Also in this case *Value* is the set of all assumed values. In analogous manner:

$$Typ : E_{atomic} \longrightarrow type$$

and

$$Typ(e) = type_e$$

In this work, we identify the event structure with its name, so:

$$e \equiv name_e$$

Function Tms:

$$Tms : E_{atomic} \longrightarrow Time$$

## 3.7 Extended functions to the primitive events

Previous functions are defined on the set of simple or atomic events, where, for example, the definition of function *Att* (and of all functions that give elements contained in the triple) is not ambiguous. It takes in input a simple event and gives the unique value of unique attribute,

$$(type_e, attribute_e, value_e)$$

If the event of interest contains more than one triple (type, attribute, value), functions previously defined have to be extended. Let us suppose a generic primitive event contained $n$ triples *(type, attribute, value)*:

$$event := (name, id\text{-}number, tms, (type_1, attribute_1, value_1), ..., (type_n, attribute_n, value_n))$$

In this case it is necessary to extend following functions as reported:

$$Typ(e, attribute_i) = type_i$$

$$Val(e, attribute_i) = value_i$$

where the functions are defined in the domain:

$$E_{primitive} \times Attribute$$

and $i$ is an integer less or equal to $n$. For primitive events let us define the function *Count*; function takes in input primitive events and gives the occurrence of triples *(type, attribute, value)* in the event.

$$Count : E_{primitive} \longrightarrow \mathbb{N}$$

So if event contains $m$ triples *(type, attribute, value)*:

$$Count(e) = m$$

Let us report the extended functions:

$$Att(e, i) = attribute_i$$

$$Typ(e, i) = Typ_i$$

$$Val(e, i) = value_i$$

where functions are defined:

$$F : E_{primitive} \times \mathbb{N}$$

and $i$ is an integer less or equal to $m$, given by *Count* that specifies position of current triple in the event $e$. Functions defined depend on primitive events and on position of triple.

## 3.8 Temporal variable

A very important question to face regards the treatment of those subscriptions depending on the time [Bib.6]. I am especially interested in explaining subscriptions declaring users interest in the variation of the value of one or more elements during time [Bib.16]. For example, let us suppose that a user is interested in the atomic event $e$ when the attribute value of $e$ increases of ten percent in at most 20 minutes. So interested user is notified when the event $e$ occurres with attribute value greater than previous ones of ten percent. This new kind of query demands to the system to store some occurred events, objects of interest of some subscriptions, until the subscriptions are satisfied and not beyond the prefixed time (of 20 minutes in the previous example). So, if the life time is not indicated in the subscription, and subscriber doesn't delete it, the system will assign a maximum time life (mtl). The time life, assigned as default, allows the system to delete events and subscriptions,

avoiding the storage of a such huge data. Let us give a simple example: we suppose the user 'anonymous' is interested to receive the notification of the occurrance of the following atomic event:

$$e = (name, id\text{-}number, tms, (type, attribute, value));$$

when a variation of element *value* is verified. The related subscription is:

$$s = (name, id\text{-}number, tme, (type, attribute \mid actual{-}value \geq previous{-}value));$$

In this case the matching system, after assigning a default maximum time of life of subscription, will make a comparison between the first *value(e)* of event published right after the time of emission of subscription and the same element *value(e)*, successfully published. In the section 3.2, I introduced the interval $DeltaT$, justifying it as the interval of publication of events. In this way events are ordered published, the interval time can be seen as time's unit. In this way the previous subscription lives in a temporal window [tme, tme + tle], so, each event $e$ published at $n$*$DeltaT$ instant (n$\in \mathbb{N}$) is considered if and only if

$$n * \Delta T < tle;$$

In this case, system matching makes a comparison:

$$value_e^{tme} < (>) value_e^{n*DeltaT+tme}$$

These considerations are to be extended to the composite events. In this case the subscriber could be interested in one or more variations of the attributes values, within specified time interval, or in the occurrence of two or more different events, within different time interval. In spite of the formalization of the composite event is reported in the next chapters, here I anticipate

that, for example, the join operator combines the sets of three events in a only one containing three t-uples. Let us remember the event is defined as a set o t-uples, while the primitive one is a set containing only one t-uple. Let us suppose that generic user is interested in the variation of one element of event or in their combination or in their intersection. For example let us suppose that user is interested in the occurrence of subsequent event obtained combained atomic ones:

$$e = (e1 \cup e2 \cup e3);$$

He wants to be notified when the value of attribute of e2 or e3 changes within m sampling:

$$att(e2)^{tme} \neq att(e2)^{tme+x} \bigvee att(e3)^{tme} \neq att(e3)^{tme+x}$$

where:

$$x <= m * \Delta T$$

If our user is interested in the combination of both events, she/he will be notified when:

$$att(e2)^{tme} \neq att(e2)^{tme+x} \bigwedge att(e3)^{tme} \neq att(e3)^{tme+x}$$

where x<m (we omitted DeltaT). Let us suppose subscriber would be notified when the attribute value of *e3* increase of $y$ quantity, within $m$ sampling, as before:

$$att(e3)(tme) = y + att(e3)(tme + x)x <= m$$

# Chapter 4

# Ennary Trees

## 4.1 Chapter's Content

In this chapter we spend some words about ennary tree structures because the important role they play in the event and subscription formalization provided, that will be clarified later.

## 4.2 Definition of Tree Structure

In computer science, a data tree structure plays an important role [Bib.55]. It allows a simple and graphical representation of hierarchical data. Data are represented with a set of nodes, the hierarchical relation between nodes can be drawn with an arc connecting a node to each of its successors . The unique node with no predecessor (no parent) is called the root of the tree. A node with no successors (no children) is called a leaf. The successors of a node are called its children; the unique predecessor of a node is called its parent. If two nodes have the same parent, they are called brothers or siblings. It is important to observe that a tree data structure can be viewed

as an acyclic oriented graph, where the hierarchical relation between node induces a partial ordered relation on the set of nodes.

## 4.3 Binary and N-ary Trees

A binary tree is a tree data structure in which each node has at most two children [Bib.20]. Typically the children nodes are called respectively left and right. A n-ary tree was born to represent more complex data structures. Generally n-ary trees are defined as structure in which exist one node with at most n children. As in a binary trees, only one node exists without any parent (root node) and each node is connected with its children or its parent trough directed edge. Let us give the following definition useful in this work:

- depth of a node n:= length of the path from the root to the node
- level of tree:= set of all nodes at a given depth
- height of a node n:= the length of the path from the node n to its furthest leaf.
- siblings:= nodes that share parent
- ancestor and descendant:= if a path exists from node p to node q, then p is an ancestor of q and q is a descendant of p.
- size of a node:= the number of descendants it has including itself.

## 4.4 Useful functions

In this section we are going to define that functions useful for visiting the tree data structure. Let us define *Node* as the set of all nodes of tree and

*Parent* as the function that map a node $n$ in its parent. *Node* set, without root, is the domain of *Parent* function, the same *Node* is the codomain for that

$$Parent(n) = p$$

for given definition:

$$Parent(root) = null$$

*Level* is defined as function that gives the node's number (a natural number) from root to current node(inclusive)

$$Level : Node \longrightarrow \mathbb{N}$$

Let us define function in details:

$$Level(root) = 1; Level(n) = Level(Parent(n)) + 1$$

In the same way, let us define tree *Depth*:

$$Depth : Trees \longrightarrow \mathbb{N}$$

where Trees is the n-ary trees set

$$Depth(tree) = max\{Level(n), n \in Node\}$$

.

Let us define *Children* and *Sibling* as correspondences between the node set and the set of node subsets

$$Children(n) = \{n_1, n_2, ..., n_k\}$$

. where n is parent of k node (where k is an integer number¿ 0).

$$Siblings(n) = \{n_1, n_2, ..., n_l\}$$

. where all l+1 node are brothers. Obviously:

$$Children(leaf) = null$$

.

$$Siblings(p) = null$$

where p can be the root node or a unique child. *Ancestor*( *Descendant*) gives the set of all predecessors (descendants) of current node.
Something more about hash table [Bib.21] is reported in appendix A.

# Chapter 5

# Composite Events and Related Functions

## 5.1 Chapter's Content

In this chapter the composite event and the related subscription are defined, starting from previous definition of primitive event. At the same time the formalization adopted is represented as tree data structure. Without loss of generality, we choose to translate the formalized event in the frame of Extensible Markup Language (the reason of such choice is explained in the following chapters). At the end of this chapter an example of a real Grid monitoring event is given, showing how it can be represented through our formalization.

## 5.2 Composite Event

A composite event can be represented with a set of t-ples of atomic events for example in following way:

$$e = (e_1, e_2, e_3, ..., e_n)$$

A composite event usually consists of primitive events connected to each other by hierarchical link. Such a composite event finds a simpler representation in the frame of tree data structure. Let us suppose a composite event, composed by 5 primitive events, is represented using meta-language XML:

```
<e1>
    <e2>
        <e3>...</e3>
        <e4>...</e4>
    </e2>

    <e5> ...
    </e5>
</e1>
```

The reported event has only one time stamps, generally written in the root t-ple. Using language of data tree structure (Ch. 3) it is recognized a root node with two children, e2 ed e5. The node e2, on its turn, has other two children called e3 ed e4. To satisfy a subscription it is necessary to visit tree data structure verifying that subscribed attributes are satisfied by respectively values.

## 5.3 Formalization of Composite Event

To represent and operate with composite events two different data structure are needed, the first is useful for representation, the second for data storage. An example is reported. Let us suppose to have a XML data tree representation [Bib.22]:

```
<bookstore>
   <book>
   <author>
               <name>Warren W.</name>
               <family-name>Gay</family-name>
   </author>
   <title>Learning linux </title>
   <home-ed>Tecniche nuove</home-ed>
   <price>40.00</price>
   <edition>
       <place>Milan</place>
       <year>1999</year>
   </edition>
   </book>
   <book>

      <author>
               <name>Wankyu</name>
               <family-name>Choi</family-name>
     </author>
     <author>
```

```
                <name>Allan</name>
                <family-name>Kent</family-name
            </author>
      <title>PHP4 - Developer Guide</title>
    <home-ed>Hoepli</home-ed>
    <price>29.99</price>
    <edition>
        <place>Milan</place>
        <year>2001</year>
    </edition>
    </book>
</bookstore>
```

To represent data in a useful manner we choose an analogous representation like that provided for primitive events (chapter 2):

$$(name\text{-}event, id\text{-}number, tms, (type, name\text{-}event, "value"), parent\text{-}of\text{-}current\text{-}event)$$

where *id-number* is a unique number, has the same meaning reported in chapter 2 (tms is reported only in a root description of composite event), and *parent* is the only parent of current event. Adopting previous formalization on the tree data, we obtain:

$$(bookstore, id - number, tms, (string, bookstore, "null"), null)$$

$$(book, id - number, (string, book, "null"), bookstore)$$

$$(author, id - number, (string, author, null), book)$$

$$(name, id - number, (string, name, "Warren"), author)$$

$$(family - name, id - number, (string, family - name, "Gay"), author)$$

$$(author, id - number, (string, author, null), book)$$

$$(name, id - number, (string, name, "Wankiou"), book)$$

$$(family - name, id - number, (string, family - name, Choi), book)$$

$$(title, id - number, (string, title, "DeveloperGuide"), book)$$

$$(home\text{-}ed, id\text{-}number, (string, home\text{-}ed, "Hoepli"), book)$$

$$(price, id - number, (float, price, "29.99"), book)$$

$$(edition, id - number, (string, edition, "null"), book)$$

$$(place, id - number, (stringplace, "Milan"), edition)$$

$$(year, id - number, (string, year, "2005"), edition)$$

Let us observe the sub-events regarding *author*, *author's name* and *author's family name*, can be substituted with the following:

$$(author, id{-}number, (string, name, "Wankiou"), (string, family{-}name, Choi), book)$$

because *name* and *family name*, are elements without children, with the same parent.

In this way it is possible to reconstruct the whole tree with the correct hierarchy. To store data we use the hash table structure useful for three fundamental operations: insertion of new data, deletion of old data, lookup of data. In this thesis we are not interested in implementing hash algorithm (appendix A), we assume to adopt the best one available, useful for our case [Bib.21]. Moreover, subscribers are notified receiving events having the form:

$$(name{-}event, id{-}number(type, name{-}event, "value"), parent{-}of{-}current{-}event)$$

they are never interested in the hash storage.

## 5.4 Subscriptions and functions defined on composite events

The functions useful to describe composite events are reported in chapter 3, there, *Parent* function, *Children* function and others were defined. Let us define now the form of generic subscription able to query composite events. In chapter 2, primitive events and relative subscription were defined as reported below:

$$e := ((name_e, tms_e, (type_e, attribute_e, value_e)))$$

$$s := ((name_s, tme_s, (type_s, attribute_s, operators_s, value_s)))$$

In defining a generic subscription let us observe a composite event is described as a finite list of n–tuples, containing all indication necessary to reconstruct original hierarchy: the parent elements. A generic subscription able to query a composite event, should contain an element name, an emission time (tme), one ore more quadruples of form:

$$(type, attribute - name, operators, value)$$

where *operator* has the same meaning reported in the chapter 2. A submitted subscription has to be matched with event structured through n-tuples, so the form of the subscription has to be faithful to the event's structure. A simple subscription on composite event is described through a list of n–tuples in number depending on the richness of information given from subscriber as reported below:

$$(name{-}event, tme{+}default{-}Time, (type, attribute{-}name, operators, value), parent(event))$$

or, if the user is interested in occurrence of nodes containing more attributes:

| |
|---|
| s:= ( name-event, tme + default-Time, (type, attribute-name, operators, value),..., (type, attribute-name, operators, value), default-parent) |

A more complex subscription, interested in occurrence of tree nodes, placed at different depth or having different parent, is expressed as a set of simple subscriptions linked by *And* operator:

s:= ( name-event, tme + default-Time, (type,attribute-name, operators, value),..., (type, attribute-name, operator,value), default-parent, default-children)

*And*

s:= ( name-event, tme + default-Time, (type, attribute-name, operators, value),..., (type, attribute-name, operators, value), parent)

## 5.5 Subscription's example on composite event

Given previous composite event, formalized in section 5.3, let us suppose to be interested in occurrence of the event book when title is *Developer Guide* and price is less then 20.00 dollars, within interval time of 3 hours. The relative subscription is:

$$(title, tme + 180, (string, title, ' equal - to', "DeveloperGuide"), book)$$

$$(price, (numerical, price, ' less - than', "20.00"), book)$$

Let us observe the element *tme + default-Time* is reported only on the first expressed tuple. If subscriber does not explicitly declare life time, a default life time is assigned.

Because elements *title* and *price* have same parent, the last two n–tuples can be substituted with:

(title, (string, title, 'equal-to', "Developer Guide"), (numerical, price, 'less-than', "20.00"), book)

where the event name is the first attribute declared from subscriber. It is possible including both simple subscription in the same t-ples. If user is interested in all events content books published by Hoepli, in the year *2002*, or published by *Hoepli* when price is less then *20.00* dollars, relative subscription becomes:

$$(home{-}ed, tme{+}default{-}Time, (string, home{-}ed,' equal{-}to'"Hoepli"), book)$$

$$(year, (string, year,' equal - to', "2002"), edition)$$

OR

$$(home - ed, (string, home - ed,' equal - to'"Hoepli"), book)$$

$$(price, (numerical, price,' less - than', "20.00"), book)$$

Both events cover the same tree, so, using a compact expression:

$$(home - ed, (string, home - ed,' equal - to'"Hoepli"), book)$$

(year, (string,year,'equal-to',"2002"), edition)

OR

(price,(numerical,price,'less-then',"20.00"), book)

Because user has not declared any temporal restriction, a default life time of subscription is assumed. After this life time subscription is deleted. In this way, the storage of enormous size of data is avoided.

## 5.6 Subscription's example applied on monitoring event

In this section, before closing the present chapter, an example of a monitoring event is reported and described. At present monitoring events are represented through tree structure using XML meta-language [Bib.24]. The meaning of terms we are going to use, for example *Virtual Organization (VO)* [Bib.25] or *Reverse Domain*, are explained later. Here we want just to treat the monitoring events as any events and rewrite them using formalization of t-ples. In the following example we treat an event reporting jobs summary related to virtual organization and site. The original document, published on INFN-GRID [Bib.26], [Bib.23] web site, was cut in the redundant parts, the real significant parts are reported below.

```
<?xml version="1.0" encoding="UTF-8" ?>

<JobSummarySiteVO xmlns="http://grid.infn.it/gridice/"
created="August 03 2006 09:52:19" Expire="2">

        <VOList>
        <VOElement>atlas</VOElement>
        <VOElement>bio</VOElement>
        <VOElement>biomed</VOElement>
        </VOList>
```

```
        <Site Name="INFN-NAPOLI">
        <Domain ReversedDomain="it.infn.na">na.infn.it</Domain>
        <Country DnsCode="it">Italy</Country>
        <VO Name="atlas">
            <Q>7</Q>
            <R>11</R>
        </VO>

        <VO Name="bio">
            <Q>2</Q>
        </VO>

        <VO Name="biomed">
            <Q>5</Q>
        </VO>

        </Site>

</JobSummarySiteVO>
```

The first line

```
<?xml version="1.0"
encoding="UTF-8" ?>
```

declares XML version used. Composite event consists of the root event named *JobSummarySiteVO*(JSSVo) published in August 03 at 9:52 (it is equivalent to the time-stamp tms, defined in our formalization), default time is assigned

with string: Expire=2. Attribute of event is :"http://grid.infn.it/gridice/".
In a compact form:

$$(JSSVO, id\text{-}number, August039:52{+}2, (string, JSSVO, http://grid.infn.it/gridice/), null)$$

$$(VOlist, id\text{-}number, (string, VOlist, null), JobSummarySiteVO)$$

$$(VOElement, id\text{-}number, (string, VOElement, "atlas"), VOlist)$$

$$(VOElement, id\text{-}number, (string, VOElement, "bio"), VOlist)$$

$$(VOElement, id\text{-}number, (string, VOElement, "biomed"), VOlist)$$

$$(SiteName, id\text{-}number, (string, SiteName, "INFN-Napoli"), JobSummarySiteVO)$$

$$(Domain, id\text{-}number, (string, Domain, "na.infn.it"), SiteName)$$

$$(Country, id\text{-}number, (string, Country, "Italy"), SiteName)$$

$$(Voname = atlas, id\text{-}number, (string, Voname = atlas, null), SiteName)$$

$$(Q, \textit{id-number}, (\textit{integer}, Q, \text{"7"}), Voname = atlas)$$

$$(R, \textit{id-number}, (\textit{integer}, R, \text{"11"}), Voname = atlas)$$

$$(Voname = bio, \textit{id-number}, (\textit{string}, Voname = bio, null), SiteName)$$

$$(Q, \textit{id-number}, (\textit{integer}, Q, \text{"2"}), Voname = bio)$$

$$(Voname = biomed, \textit{id-number}, (\textit{string}, Voname = biomed, null), SiteName)$$

$$(Q, \textit{id-number}, (\textit{integer}, Q, \text{"5"}), Voname = biomed)$$

Our monitoring event is represented by 15 n–tuples. Let us observe where the attribute value is null (for example for event named *Vo name=atlas*), *type* has value *string*. when attribute value is null, the default type could be *string* or null. This reported example is the most nested and complex event produced by actual monitoring service. It is perfectly represented through our formalization.

## 5.7 Subscriptions and hash data structure

It is necessary to underline that in this work we are not interested in the storage of data in a hash structure. We suppose current subscription is translated in a useful form for the lookup phase.[Bib.21]

# Chapter 6

# The Semistructured Data and The Extensible Mark-up Language

## 6.1 Chapter's content

The formalization of events and subscriptions, described in the previous chapters, falls in the *schema-less* or *self-describing data* theory, also called *semi-structured data* theory. In this chapter the definition and some fundamental notions inherent to semistructured data are reported; moreover an explanation of this choice instead of structured data is given.

## 6.2 Structured and Semistructured Data

Data without any kind of structure are often called schema-less data or self-describing data [Bib.27]. Both terms underline there is no separation between the description of the type of data and its instance. Generally when a piece

of data is programmed, first the relative structure is described (schema and type) and then the instance of that type, built in that schema, is created. On the contrary, semistructured data are directly described using a simple and intuitive syntax, where description combines both type and structure, or, sometimes, type, structure and instance, as reported in the third example. Example: atomic event

$$event := (name, tms, (type, name - attribute, value))$$

$$(thermodynamic, 12:31, (float, pressure, "2.0"))$$

combining both expressions:

```
(name= thermodynamic, tms= 12:31,
(type= float, name-attribute= pressure, value=
"2.0")
```

This representation is based on label-value pairs expressed using n–tuple-like structure. Data can be graphically represented as a graph where given events are nodes connected by edges to their values(fig.1); let us observe edges are characterized by no terminal names.

Generally semistructured data are graphically represented though graphs theory. In this work the generic event and generic subscription formalized can be represented through trees theory, as described in chapter 3. Let us explicitly observe trees set is a subset of graphs set, with stronger properties.

In programming languages definition of types and data structure is very rigid. A small change in the data requires a revision of all structure defined. The chosen philosophy is able to explain our data and lets the data describe them self. This kind of information takes *self describing data* name.

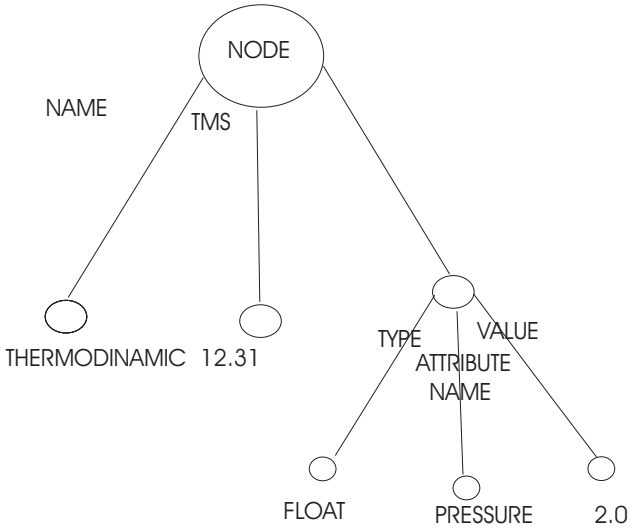


Figure 6.1: HTML representation

## 6.3 The Reason of Choice of Semistructured Data

The choice of semistructured data is necessary where information are not consistent to conventional model, where data intrinsically have no structure, for example on the Web. Users sometimes users have to compare information, coming from different sources, described using different format (for example produced by different databases). In this case any structured language produces a rigid representation that is unable to describe some changes in the data. Of course, *self describing data* wastes space, if naively stored, specially if many descriptions with each data item are needed. It is important to observe that data we treat, have a very short life so, any lost of space during storage is compensated by frequent deleting of information. On the other hand, information are all preserved and expressed using a very simple syntax, near to natural language.

## 6.4 XML as Semistructured Data Model

XML is a consolidate standard for data exchange on the Web. In this section we want to introduce this meta-language underlining how it is adaptable to the description of semistuctured data. We are not going to give a full description of XML, our intention consists of disclosing XML properties, emphasizing its role as a data exchange format. The necessary of common and widespread language of data exchange useful to combine information coming from different source (usually explained with languages having different syntax), finds in the XML meta-language a universal data exchange format. XML may be seen as a tool to develop languages describing content of information (rather then presentation), that is able to control the syntax and semantic information aspect. It permits to define new nodes (where node have the meaning reported in the previous section) or to nest news to arbitrary depth. The basic XML syntax is perfectly suited for describing semistructured data.

### 6.4.1 XML and HTML: the differences

It appears important at this point to underline HTML can be seen as an "instance" of XML but it can not be confused with it. HTML is the most popular language for describing Web pages. It consist of tag fields such as

```
<p> ... </p>  containing text, for example:

<p> <b> Carmen </b>, <b> Santiago </b>,
    <b> 1978-Agoust-15 </b>, <b> Las Palmas </b>,
    <i> CarSantiago@laspalmas.es </i>
</p>
```

where means a new paragraph is beginning (at contrary means a paragraph is finished), <b >and <i >mean that the text contained may respectively appear bold and italic.

HTML was specifically designed to describe data presentation and not its content. On the contrary, XML generates languages able to describe content [Bib.22]. Trough XML rules it is possible to define new tags, made "ad hoc", for specific purpose (for example in the mathematical, chemical or physical frame, including in the document a description of used grammar), to indicate specific structures. Moreover it is possible nesting structures to arbitrary depth.

For example:

```
<bookstore> ... </bookstore>
```

defines a new structure named *bookstore* that can contain other substructures like *books* and so on. At the end, let us observe XML provides no instructions about its displaying. These information are generally reported in a *style-sheet* (a document developed in the homonymous language able to translate XML data in HTML that allows displaying by any browsers). Essentially, XML allows syntax and semantic data transmission, for this reason it has become a major standard for data exchange between different applications.

### 6.4.2 Basic Syntax

Let us now show some basic components of XML meta-language. The real basic component in XML is the *element*, a piece of text contained between

start-tag and end-tag. Let us observe tags in XML are not predefined as in HTML, generally tags are defined by users. In the following example

```
<student>
 <name> Carmen </name>
 <family-name> Santiago  </family-name>
 <birth-date> 1978-Agoust-15 </birth date>
 <birth-place> Las Palmas </birth-place>
 <email> CarSantiago@laspalmas.es</email>
</student>
```

the structures between the tags are called *content*, so the *name, family name, date, place of birth and email* are the content of *student*, moreover each tag contained in the *student* tag is its *sub-element*. On the contrary of HTML description, here it is underlined the content of data information, but no indication about displaying is given.
Another important "object" of XML is the the *attribute*. In this context the term *attribute* means *property* and it is expressed using name-value pair. The Extensible Mark-up Language allows to associate the elements with attributes, so the previous example of the structure named *student*, can be described in the following way:

```
<student name="Carmen" family-name="Santiago">
<birth-date>1978-Agoust-15 </birth date>
 <birth-place> Las Palmas </birth-place>
 <email> CarSantiago@laspalmas.es</email>
</student>
```

where name= "Carmen" and family-name="Santiago" are two pairs name-value (two attributes), associate with *student* element. Also attributes are

arbitrarly defined by users.
When tags nest properly and defined attributes are unique, the XML document is *well formed* that means XML data will parse into a labeled tree.

## 6.5 From XML to Semistructured Data Representation

The XML syntax is suitable to represent semistructured data. In the previous section an instance of structure *student* was reported, let us now analyze the same structure as a semistructured data, translating it through our formalization. The following XML structure

```
<student>
<name> Carmen </name>
<family-name> Santiago  </family-name>
<birth-date> 1978-Agoust-15 </birth date>
<birth-place> Las Palmas </birth-place>
<email> CarSantiago@laspalmas.es</email>
</student>
```

can be seen as a composite event where the event *student* is the root event, while substructures are all siblings and children of the root. The semistructured formalization has the subsequent representation:

$$(student, id - num1, tms, (null, student, null)null)$$

$$(name, id - num2, (string, name, "Carmen"), student)$$

$(family-name, id-num3, (string, family-name, "Santiago"), student)$

$(birth-date, id-num4, (alpha-num, birthplace, "1978-Agoust-15"), student)$

$(birth-place, id-num5, (sting, birth-place, "LasPalmas", student)$

$(email, id-num6, (alpha-num, email, "CarSantiago@laspalmas.es"), student)$

where composite event contains 6 primitive events, reporting their identification number.

If users express the same XML structure using more attribute elements, for example:

```
<student name="Carmen" family-name="Santiago">
<birth date="1978-Agoust-15" place="Las Palmas" </birth>
<email> CarSantiago@laspalmas.es</email>
</student>
```

the semistructured formalization becomes:

```
(student, id-num-i, tms,(string, name,"Carmen"),
(string,family-name, "Santiago"), null)

(birth, id-num-i+1, (alpha-num, birth-date,"1978-Agoust-15"),
(string birth-place,"Las Palmas" ), student)
```

```
(email, id-num-i+2, (alpha-num, email,"CarSantiagolaspalmas.es"),
student)
```

that is still a composite event but more compact, where the user has used the syntax element *attribute*, and has introduced another structure named *birth* enclosing birth-date and birth-place.

## 6.6 Translation Function

It appears necessary to underline that generally schema-less data are unordered. On the contrary XML representation is based on ordered structure, the exchange of two any structures, nested at same level, produces two different documents. This aspect (and some others) is not easily reconciling with semistructured data theory but formalization we developed, produces ordered events, using ordered t-ples.
Let us now observe the transformation from document, represented trough our formalization, to XML document (and *vice-versa*) can be realized introducing a translation function that associates a simple node of XML tree to a primitive event. The function, named T, has the following input:

$$(name, tms, id - num, (type, atrribute, value)$$

and gives the following output

$$(nametms = ""idnum = "", attribute = "value")$$

In a analogous manner it can be extended to composite event. The idea we want to underline is that formalization realized is not dependent on the language used.

## 6.7 Query Language for Semistructured Data

There is a deep difference between accessing data from database or from document. If the data of our interest are stored in a database it is necessary to elaborate a right query to obtain it. If we need data contained in a Web document, we provide a URL (Uniform Resource Locator) and obtain, as answer an HTML page. HTML page is usually obtained on the base of a query created by user interface. But, while query can give a detailed answer, an HTML page represents the document where it is necessary to search for the answer. In this context emerges the importance of a query language for Web data, in general, for semi-structured data. Obviously we want our query language to be powerful and able to use complex predicates, we already want it to combine and perform data. Here we describe some general points that are necessary for a query language, independently from specific format of data. It should be expressive, it means it should be capable of expressing all operations of relational algebra. It should be characterized by a clear semantics to allow query transformation. It should be composed, where composed means the output produced from language should have the same form of answered data.

## 6.8 Query Language for XML: XPath

In this thesis we choose to query data represented in XML, with XPath language, which in the last years has become a W3C standard. XPath is, at present, one of the most popular language for query XML documents [Bib.28]. It can be explained as a syntax for defining parts of an XML document. It is based on the usage of path expressions to navigate in XML documents and to select nodes of interest. These path expressions are very similar to those

represented by a traditional computer file system, this characteristic allows users to approach its philosophy.

# Chapter 7

# The Pseudo-code: An Idea of Algorithm

## 7.1 Principal Procedures for primitive events

In this section we describe, through the pseudo-code, the algorithms that allow users to query published events in order to verify subscription's satisfaction. Let us begin analyzing a primitive event. The form of a primitive event and of generic a subscription, ignoring the id-number, is:

$$e := ((name_e, tms_e, (type_{e1}, attribute_{e1}, value_{e1}), ..., (type_{en}, attribute_{en}, value_{en})))$$

$$s := ((name_s, tme_s, (type_s, attribute_s, operators_s, value_s)))$$

Previous subscription is interested in all events named $name_s$, published within temporal interval

$$[tme_s, tme_s + defaultTime]$$

where:

$$\exists i \in 1, ..., n$$

for that

$$type_s = type_{ei}$$

$$attribute_s = attribute_{ei}$$

and

$$operators_s(value_s, value_{ei}) = True$$

I create suitable procedures that implement the matching phase:

```
Procedure Base_Validation(s,e)
 Base-validation= False
 if name_{s}== name_{e}
        and
    tme_{s} < t <tme_{s}+ Default-Time
    Base-validation= True
 end-if

Procedure Base_Matching(s,e)
 Base-matching= False
 if type_{ei}=type_{s}
          and
    attribute_{ei}=attribute_{s}
          and
    operator_{s}(value_{ei}, value_{s})= true
```

```
     Base-matching= True
end-if
```

where *Base-Validation* verifies if field *name* of published event matches *name* of one of submitted subscriptions; at the same time the procedure controls the compatibility in terms of time. If both conditions are satisfied, *Base-Matching* procedure is invoked. Last called procedure verifies type's and attribute's identity of the event and subscription; the procedure also verifies the operator satisfaction. Both procedures are called in the procedure *match*, in the following way:

```
match(s,e)
    begin
      Base_Validation(s,e)
      if Base_Validation(s,e)=true
        for i =1,..., Count(e)
            Base_Matching(s,e)
        end for
        if Base_Matching=true
            Notify_subscriber,
        end-if
      end-if
    end
```

The function *Count*, defined in the chapter 3, gives the number of triples contained in the event n–tuple. The procedure *Notify-subscriber* takes the matched event and, on the base of id-subscription number, sends it to all subscribers that expressed their interest.

# Chapter 8

# The Recent Emerging Technology: Computational GRIDs

## 8.1 Chapter's Content

In this chapter the computational Grids are introduced underlining that the importance of Grid concept is caused by concrete and specific problem of large scale resource sharing. Then the *Grid problem* and the *Grid infrastructure* are defined explaining the differences with the others existent technologies. At the end of this chapter an important aspect of the *Grid problem*, object of this work, is described and analyzed: the *Grid monitoring*

## 8.2 The Grid History

The term "Grid", coined in 1998, in analogy with the electric power Grid, identifies "a proposed distributed computing infrastructure for advanced sci-

ence and engineering" [Bib.30], [Bib.26]. The "electric power grid metaphor" is sometimes used to explain the revolutionary aspect of computational Grids. In their book "The Grid", Ian Foster and Carl Kesselmann make a comparison between current status of computation and that of electricity around 1910, underlining that "the truly revolutionary development was the electric power grid and the associated transmission and distribution technologies" rather than electricity discovery (Chapter 2) [Bib.29]. When people use electricity they do not need to know how and where it is produced, they do not need to know which kind of technologies were developed to transmit and to distribute it to everyone, they just plug their device in the wall socket and use it [Bib.31]. At present millions of computers, connected through internet and sharing information, populate the planet. In this scenario the science dream of using unlimited computational power seems to become reality [Bib.25]. "What we need is an infrastructure and standard interface able to providing transparent access to all this computing power and storage space in a uniform way" [Bib.31]. Across the net the end user will not see any more the many connected machines but he will just submit his request and relative requirements and the built Grid will satisfy him. The Grid infrastructure will find and allocate suitable resources, will monitor the running process and finally will notify the user sending him the final result. Since the described process is perfectly analogous to that of electrical power we think it will produce the same social economic growth in the society.

## 8.3 The Grid and Other Technologies: Analogies and Differences

Nowadays The Grid infrastructure has developed a great deal and the Grid term now is used to denote all that is related to "advanced networking". It appears necessary to underline that there is a deep difference between this emergent technology and other major ones, such as distributed computing, peer to peer computing or internet technology, but the progress made in the Grid frame produced significant benefits also in all the other fields. We define Grid infrastructure as a set of coordinated systems sharing [Bib.32]. We define Grid problem as "flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions and resources" (also called VO, Virtual Organizations). It appears necessary to spend some words to define the Virtual Organization, which represents the principal reason of Grid development . Let us suppose people, various institutions or both want to permit direct access to their software, data and resources in the range of collaboration that defines priorities of each component and the sharing conditions. The group defined by this conditions is called Virtual Organization [Bib.25]. Sometimes such organizations differ enormously in the nature of their problems, aims, size of data, duration of collaboration and specially in the sociology of involved parts. But it is possible to recognize some common requirements that find their solution in the Grid philosophy. In fact all VOs need flexible sharing modality, ranging from *client server* to *peer to peer* model [Bib.34], [Bib.33], [Bib.54], a VO always includes components working in different places that want to share remote resources, in the respect of assigned priorities without any loss in terms of security and quality of services. The latter requirements are not satisfied by current distributed

systems. The same Internet technology allows communication and information exchange between computers but it isn't able to provide sharing access to multiple resources with, for example, the computational purposes. During last years many distributed computing technologies were born as, for example, the *Open Group's Distributed Computing* that assures the resources sharing among different organizations in the respect of security of all involved parts [Bib.25]. This technologies results too inflexible, not adaptable to VO requirements. Other technologies permit the access only to some resources types. So, each technology can at most satisfy some well defined demands. In this context the real importance of the Grid technology emerges. At present, protocols, services and tools satisfying VO's requirements are available and Grid technology is integrated, rather then competing, with other distributed technologies.

## 8.4 Grid Architecture: a Short Description

In this section, Grid technology is described under the architecture point of view [Bib.35], recognizing principal system components, describing their function and the rules of interaction. The first requirement to be satisfied is the interoperability that can be translated in common protocols. Interoperability allows participants, although different programming languages, platforms, and programming environments, to start sharing relationship. These common protocols permit resources sharing to VO users (including priority rules). A standard based open architecture, common protocols, *Application Program Interface* and Software Development Kit constitute the Grid middleware. In the following description Grid architecture is organized in layers. Components of each layer are built on the base of functions provided by lower

layer. In this description we have 5 layers: Fabric(the lowest), Connectivity, Resource, Collective and Application. The number of protocols defined for Resource and Collective have to be small, this two layers allow the sharing of individual resources. To better understand this high level description it is necessary to analyze the protocols defined and tested in many projects [Bib.36], [Bib.37], [Bib.38], [Bib.39], [Bib.40] .

## 8.5 The Grid Monitoring

Monitoring activity is essential to the management of a Grid System [Bib.43], [Bib.50]. The Grid Monitoring consists of measuring significant Grid resource related parameters [Bib.41], [Bib.42] with the aim to describe behaviour, performance and usage of Grid System. Development of adaptable Grid monitoring system has to deal with a new class of problems as geographical distribution of the resources or diversity of involved parts. In Grid environment we can distinguish two monitoring phases: infrastructure monitoring and application monitoring. The first regards collecting information about Grid resources in order to build the grid resources history. The second phase, on the base of collected information, permit to satisfy users demands.

### 8.5.1 Terms and Concepts

In this section the definition of terms inherent Grid monitoring activity is given. In the Grid monitoring context with the *entity* term we define each monitored object. We define as *attribute* a characteristic of a generic *entity* [Bib.41]. With the term *measure* we describe a procedure which assign to each attribute a value, on the base of observed phenomena. We define *measure unit* an adopted quantity for convention. The measure de-

finition implies necessarily the presence of sensors able to evaluate it. We define *sensor* as a monitoring entity process with the aim to produce related *observation.*[Bib.41]

### 8.5.2 Grid Monitoring Phases

In the Grid Monitoring process it is possible to distinguish four main phases: (1) the *generation* where sensors query entities and produce the values related to the obtained measures (2) *distribution* where obtained information is distributed to the suitable components (3) *filtering* that permits to choose only the useful information on the base of fixed criteria (4) *presentation* where information are distributed to interested users for management phase. Users interested in the monitoring information are distinguished in three categories: Grid Operators, Virtual Organization managers and Site Administrators. They present same common requirements, so they can satisfied from a integrated monitoring tools. The the most common important aspect among different categories is the choice of measurements to be performed. Only a small subset is specifically designed for each of them [Bib.58].

## 8.6 INFN-GRID: Naples's project

The INFN is involved in the LHC (Large Hadron Collider) project [Bib.44], [Bib.48], [Bib.47], under construction at CERN and expected to provide first collision in 2007. The high-energy physics experiments at the LHC will collected data for 15 years. The resource requirements are so large that they are not provided in a single geographic location, they need a Grid infrastructure and a adaptable monitoring service [Bib.45]. The INFN, section of Naples, is involved in the design and development of such monitoring service

since 2001. In these years the monitoring team produced a monitoring tool: GridICE [Bib.23]. A lot of Grid Projet (Russian Grid, South Eastern European Grid, Europe and Latin America Grid, EuMed Grid, EuChina Grid) [Bib.24], [Bib.46] chose GridICE as a monitoring tool. The characteristics of this tool are reported in appendix.

# Conclusion

In this thesis we have described general lines of monitoring in distributed systems, with regards to the grid monitoring. We adopted the publish/suscribe paradigm with content-based communication model. In this context, interests of users in the occurrence of some events and the variation of some contained attributes (during time), has been formalized. In this thesis atomic, primitive and composite events, simple and complex subscriptions have been structured; the functions that take events set as input and give contained attributes, have been defined. This formalization, structured independently from the monitoring problem, has been then adopted to the grid monitoring. It has been demonstrated that realized formalization is able to describe each monitoring event currently produced and allows users to submit expressive subscriptions.

It is necessary to underline that the model shown is applicable to other contexts characterized by publish/subscribe paradigm, such as Economy or Physics or Earth observation, moreover it is defined independently of language used. This formalization finds its natural positioning in the future Grid Monitoring Service under construction.

# Appendix A

# Algebra of events

It is interesting observing some mathematical properties simply analyzing formalized events and subscriptions through language of the Algebra. Let's define set $E$ containing all published events, ($e \in E$), let's define set $S$ containing all submitted subscriptions, $s \in S$. Given $s$ subscription It is simple to observe that set of events satisfying $s$ is a subset of $E$, Let's indicate it $E_s$.

$$E_s = e \in E, \mid e \sqsubset s$$

If an event satisfies a given subscriptions it covers the subscription. The covering notion introduced is a matemathical relation. We say $s \sqsubset e$ se $e \in E_s$, or $s \sqsubset s'$ if

$$E_s \subseteq E_{s'}$$

Covering relation on the set subscriptions is a no total ordering relation, induced by set inclusion.

# Appendix B

# Hash Tables

A hash table is a data structure that associates keys with values. This structure supports efficiently a lookup operation: given a key (for example a personal code), it finds the corresponding value (e.g. actual address ). The concept of hash table is strictly related to the concept of hash function. It works by transforming the key using a hash function into a hash, that is a number used to locate and to find the desired value. The lookup on average is constant-time O(1) , regardless of the number of items in the table. However, in the worst case, lookup time can be as O(n). Compared to other data structures, hash tables are more useful when it is necessary to store a large numbers of data records.

## B.1 Collision Resolution

If two keys hash to the same index, the corresponding records cannot be stored in the same location. So, if it's already occupied, it is necessary to find another location for the new record, choosing a criteria so that it's easy to find it later. There are a number of collision resolution techniques, but

the most popular are chaining and open addressing.

## B.2 Hash collision resolved by chaining

In the simplest chained hash table technique, each slot in the array references a linked list of inserted records that collide to the same slot. Insertion requires to find the correct slot and appending to either end of the list in that slot; deletion requires to search the list and removal.

# Appendix C

# GridICE: A Monitoring Tool

As we said, in the Grid computing monitoring activity plays an important role. In this section the GridICE tool is described, analyzing better the notification service [Bib.23], [Bib.24], [Bib.41], [Bib.42], [Bib.49].

## C.1 GridICE: the architecture

The GridICE architecture is developed in five layers, including the producers of monitoring data and the final consumers of these data

### C.1.1 The layers

- Measurement Service: the first layer is the Measurement Service, whose goal is to probe resources for simple or composite metrics. The gathered data are locally stored in a site repository.
- Publisher Service: the second layer is the Publisher Service. This layer organizes the gathered data to potential consumers.
- Data Collector Service: the third layer is the Data Collector Service.

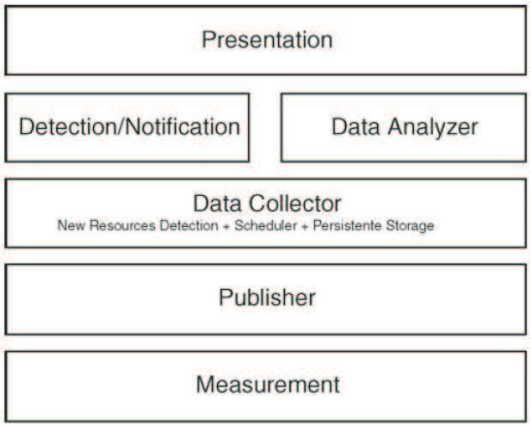


Figure C.1: The GridIce Architecture

It permits the collection of historical monitoring data and is developed in several components. The main ones are the New Resources Detection component, the Scheduler component, and the Persistent Storage component.

- Detection/Notification and Data Analyzer Services: the fourth layer comprises two services. The Detection and Notification Service provides a flexible and configurable means for event description, detection, and notification. This service has to be able to: (1) send timely notifications using different communication services; (2) escalate through different levels of notification if no actions are taken (3) help in the diagnosis of notified events by enabling the browsing of historical information at various levels of aggregation and detail; (4) adapt automatically to the dynamics of the resource part of the virtual pools accessible

to VOs. The second service of this layer is the Data Analyzer Service. It provides performance analysis, usage level, and general reports and statistics. It can be configured to generate and send periodical reports of Grid activity, and possibly also Grid structure.

- Presentation Service: the last layer of the architecture is the Presentation Service, a web-based graphical user interface that offers a view of monitoring information.

## C.2 GridICE Implementation

This architecture was implemented with component software. In the first layer, the sensors for all the defined metrics in the extended GLUE Schema was developed and tested. In GridICE previous version, the data collector used by default was the Lemon tool. GridICE can dialogue with different local monitoring tools.

In the second layer, Globus MDS Version 2, the information service usually adopted by large-scale Grids in High Energy was adopted. Because of Important advancements in the area of content-based publish/subscribe overlay networks and XML-based technologies, the information service will change. For layer three, the scheduling of observations is based on Nagios For layer four, simple versions of the Detection/ Notification and the Data Analyzer Services have been implemented, but both still require improvements to be used in production environments. For layer five, the presentation service relies on a web interface written in PHP.

## C.3 GridICE: Usage and Results

Usage and results of GridICE monitoring tool are related to the testing and production phases of Grid systems related to the LHC experiment. LHC (Large Hadron Collider) is the world's largest and most powerful particle accelerator currently under construction at CERN, the European Organization for Nuclear Research. GridIce monitoring tool took place in the context of one of the four LHC experiments: the Compact Muon Solenoid [Bib.23]. This experiment previews a wide and international collaboration, involving more than 2,000 people coming from 160 organizations in 36 countries. One of its goals is to confirm the existence of the Higgs boson (predicted by the Standard Model but not detected so far by any experiment). The second large deployment of GridICE is within the Italian Grid infrastructure, managed by the Italian Institute for Nuclear Physics (INFN). It consistes of more than 20 sites among the most important Italian universities with about 1,500 CPUs and more than 15 Terabytes of disk space. The third large deployment of GridICE is its integration in LCG middleware release 2. The goal of LCG is to fulfill LHC computing needs by deploying a worldwide computational Grid service, integrating the capacity of scientific computing centers spread across Europe, America and Asia. The LCG system includes more than 70 sites in the world with about 6,000 CPUs (35 Terabytes of disk space).

# Appendix D

# The Events Published by GridICE

In this section classes and characteristics of Grid monitoring events are reported [Bib.59]. The event of interest are classified in three types:

- Site Events
- Gris Events
- Host Services Events.

At present GridICE publishes the events of interest on a web page and in a XML format.

## D.1 Site Event

This class of event gives information about computing and storage resources of specific site monitored by GridICE . The generic event is characterized by:

- number of Computing Elements

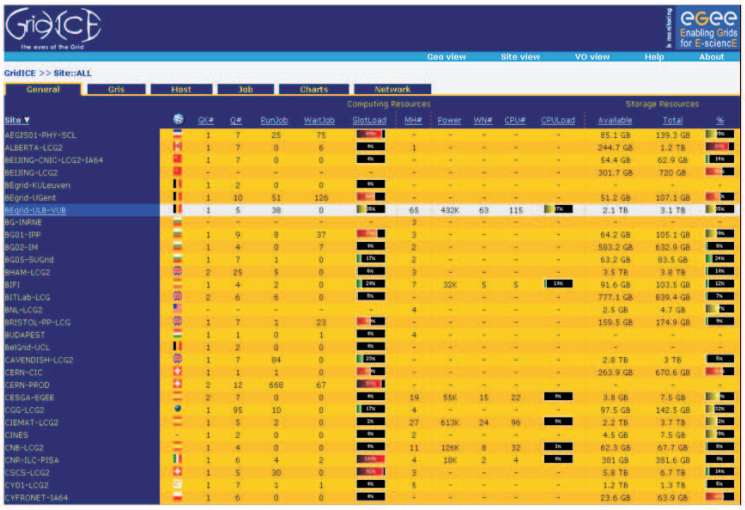


Figure D.1: GridIce web site

- total job slots
- available job slots
- percentage of used job slots
- number of available gatekeepers
- running job
- waiting job a
- percentage of running jobs respect to computing resource
- monitored hosts
- computing power
- available nodes

- total CPU
- CPUs to WNs.

The grid monitoring events generally contain information about Storage Elements such as the disk space available, total space, and the percentage of disk space used [Bib.58] .

A Site Event can be represented in XML, as following reported

```
<SiteData Name="Site Name">
     <Country DnsCode="code">Country Name</Country>
     <QueuesNum>1</QueuesNum>
     <SlotsNum>2</SlotsNum>
     <FreeSlots>2</FreeSlots>
     <SlotsLoad>0</SlotsLoad>
     <GateKeeperNum>1</GateKeeperNum>
     <RunningJobs>0</RunningJobs>
     <WaitingJobs>0</WaitingJobs>
     <JobsLoad>0</JobsLoad>
     <MonitoredHosts>-</MonitoredHosts>
     <BogoMips>-1</BogoMips>
     <WorkingNodesNum>-1</WorkingNodesNum>
     <CPUNum>-1</CPUNum>
     <CPULoad>-1</CPULoad>
     <StorageAvailable>616144</StorageAvailable>
     <StorageUsed>759135</StorageUsed>
     <StorageTotal>1375279</StorageTotal>
     <StorageLoad>55</StorageLoad>
</SiteData>
```

### D.1.1 An example of subscription

If a subscriber is interested in the occurrence of Site type event, she/he will expresses her/his interest in one or more attributes described: For example users could be interested in receiving notifications if:

- The Queues inherent to CERN-CIC site are more than 10 and no working node is available.
- There are many sites without storage element available space.

## D.2 Gris Event

This kind of event describes the Gris. It expresses grid host names, the version of installed middleware domain name, the country, the name of administration site, the gris type among Computing Element, Storage Element, Extended Gris or a BDII. An example of gris event is following reported:

```
<GRISElement URI="ldap://host:port/mds-vo-name=local,o=grid">
      <HostName>Host Name</HostName>
      <Middleware>LCG-2_6_0</Middleware>
      <Domain ReversedDomain="it.infn.na">Domain Name</Domain>
      <Country DnsCode="...">...</Country>
      <SiteName>Site Name</SiteName>
      <Type Code="4">EX</Type>
      <LastCheck UnixTime="1131844923">2005-11-13 02:22</LastCheck>
      <ConnectionCode>1</ConnectionCode>
      <Entries>0</Entries>
      <ConnectionCodeSince UnixTime="1131726743">
            2005-11-11 17:32
```

```
      </ConnectionCodeSince>
      <Scheduling>2</Scheduling>
      <IDRes>668</IDRes>
      <Contact>mailto: mail adddress</Contact>
</GRISElement>
```

Users interested in a gris event could submit their interest in the occurrence of event where connection code is OK.

## D.3 Host Service Event

This kind of event gives information on the most relevant hostes in the Grid. An example of Host Service Event is following reported:

```
<Site Name="Site Name">
     <Domain ReversedDomain="...">Domain Name</Domain>
     <Country DnsCode="...">Country Name</Country>
     <Role Name="All">
          <Total>0</Total>
          <KO>0</KO>
          <Dis>0</Dis>
     </Role>
     .....
     <Role Name="Others">
          <Total>0</Total>
          <KO>0</KO>
          <Dis>0</Dis>
     </Role>
</Site>
```

# Bibliography

[Bib.1] Yigal Hoffner *Monitoring in Distributed Systems*, Report Ansa Phase III, 1994.(www.ansa.co.uk)

[Bib.2] DOMAINS Standardization, Document D2f V1.0, *Distributeed Open Management Architecture in Networked Systems*, Esprit Project No 5165, (1992).

[Bib.3] LaBarre *Management by Exception: OSI Event Generation, Reporting and Logging*, The MITRE Corporation, 2nd IFIP Symposium on Integrated Network Management, Washington, USA (1991).

[Bib.4] McDowell, Helmbold, *Debugging Concurrent Programs*, ACM Computing Surveys, 21(4), pag. 593-622 Dec. 1989.

[Bib.5] Joyce, Lomow, Slind, Unger, *Monitoring Distributed Systems*, ACM Transactions on Computer Systems, 5(2), 121-150 May 1987.

[Bib.4] Sloman M.,*Policy driven managment for distributed systems*Journal of Network and Systems Management, Springer, 1994

[Bib.5] Masoud Mansouri-Samani *Monitoring of Distributed Systems*, University of London, Imperial Colloge of Science,Technology and Medicine, Department of Computing, 1995

[Bib.6] Masoud Mansouri–Samani and Morris Sloman *GEM A Generalised Event Monitoring Language for Distributed Systems* EE/IOP/BCS Distributed Systems Engineering Journal Vol. 4, No. 2 June 1997

[Bib.7] Mansouri–Samani M., Sloman M. *Monitoring Distributed Systems* Network, IEEE, Vol7, Issue 6, pag. 20-30, 1993,

[Bib.8] Bates P. *Debugging Heterogeneous Distributed Systems Using Event-Based Models of Behaviour*, ACM Transactions on Computer Systems, Vol. 13, No. 1, pp. 1–31, (1995)

[Bib.9] Wolfson, Sengupta, Yemini, *Managing Communication Networks by Monitoring Databases*, IEEE Transactions on Software Engineering, Vol. 17, No. 9, pp. 944–953, (1991).

[Bib.10] [en.wikipedia.org/wiki/System]

[Bib.11] [www.ichnet.org/glossary.htm]

[Bib.12] Eugster, Felber, Guerraoui, Kermarrec, *The Many faces of Publish/Subscribe*,ACM Computing Surveys, Volume 35 , Issue 2 table of contents, Pages: 114 - 131, 2003

[Bib.12] Fidler, Jacobsen, Mankowski, *The PADRES Distributed Publish/Subscribe System*, Features Interaction in Telecommunication and Software System, page 13, IOS press 2005.

[Bib.13] Ceccanti, Panzieri, *Content–Based Monitoring in Grid environments* Proc. 13th. IEEE, International Workshop on Enabling Technologies.2004

[Bib.14] Aguilera, Strom, Strurman, Astley, Chandra, *Matching events in a Content-based Subscription System*, Proc. 18th. Annual ACM Symposium on Principles of Distributed Computing, pg. 53–61, 1999.

[Bib.15] A. Carzaniga, A. L. Wolf. *Contentbased networking: A new communication infrastructure*. In NSF Workshop on an Infrastructure for Mobile and Wireless Systems, Scottsdale, Arizona, 2001.

[Bib.16] A.Carzaniga, A.L.Wolf *Content-based Addressing and Routing: A General Model and its Application*, Proc. NFS Workshop on an Infrastructure for Mobile and Wireless Systems.

[Bib.17] A. Carzaniga, M.J. Rutherford, A.L. Wolf, *A Routing Scheme for Content-Based Networking*, INFOCOM 2004, 23th Annual Joint Conference of IEEE Computer and Communications Societies, Vol. 2, pag. 918–928

[Bib.18] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. *Design and evaluation of a widearea event Notification Service*. ACM Transactions on Computer Systems, 19(3):332–383, 2001.

[Bib.19] A. Carzaniga, A. L. Wolf. *Forwarding in a contentbased network*. SIGCOMM '03, Karlsruhe, Germany, Aug. 2003.

[Bib.20] Aho, Ullman *Fondamenti di iInformatica*, Zanichelli, 1994

[Bib.21] *Secure Hash Standard*. U.S. Department of Commerce/NIST, National Technical Information Service, Springeld, VA, Apr 1995.

[Bib.22] Erik T. Ray, O'reilly *Learning XML*.

[Bib.23] Andreozzi, De Bortoli, Fantinel, Ghiselli, Rubini, Tortone, Vistoli, *GridICE: A Monitoring Service for Grid Systems*, Future Generation Computer Systems, Elsevier (www.sciencedirect.com).

[Bib.24] grid.infn.it/gridice

[Bib.25] I. Foster, C. Kesselman, S. Tuecke. *The Anatomy of the Grid, Enabling Scalable Virtual Organizations*,International Journal of High Performance Computing, Vol. 15 n. 3, 2003.

[Bib.26] I. Foster, C. Kesselman *The Grid 2*. ELSEVIER, 2004.

[Bib.27] S.Abiteboul, P.Buneman, D.Suciu. *Data on the Web: from relations to semistructured data and XML*, Morgan Kaufmann Publishers, 2000.

[Bib.28] www.w3.org

[Bib.29] I. Foster, C. Kesselman, M. Kaufmann *The Grid: blueprint for a new computer infrastructure*, 1998 Publisher Inc. San Francisco.

[Bib.30] K. Czajkowki, S. Fitzgerald, I. Foster, C. Kesselman *Grid Information Service for Distributed Resource Sharing* Proc. 10th IEEE, International Symposium on High Performance Distributed Computing 2001.

[Bib.31] http://web.datagrid.cnr.it/LearnMore/LearnMore4.jsp.

[Bib.32] I. Foster, C. Kesselman, J.M. Nick, S. Tuecke *The Phisiology of the Grid: An Open Service Architecture for Distributed Systems Integration*, CiteSeer 2002.

[Bib.33] the Overlay network, Spotlight *Overlay Networks: A Scalable Alternative for P2P*, IEEE Internet Computing, July-August 2003.

[Bib.34] I. Stoica, R. Morris, D. Karger, M.F. Kaashooek, H.BalaKrishnan *Chord: A scalable Peer to Peer Lookup Service for internet Applications*, Mit Laboratory for Computer Science

[Bib.35] I. Foster, C. Kesselman, J.M. Nick, S. Tuecke *Grid Service for Distributed System Integration* Computer, Vol. 35, Issue 6, pag. 37–46 2002.

[Bib.36] I. Foster, C. Kesselman, *Globus Project: A status Report*, Heterogeneous Computing Workshop, IEEE, 4-18, 1998

[Bib.37] Johnston, Gannon, Nitzberg, *Grids as Production Computing Environments*, The Engineering Aspects of NASA's Information Power Grid, Proc. 8th IEEE International Symposium on High Performance Distributed Computing, 1999

[Bib.38] Beiringer, Johnson, Bivens, Humphreys, Rhea, *Constructing the ASCI Grid* , Proc. 9th IEEE International Symposium on High Performance Distributed Computing, 2000

[Bib.39] www.griphyn.org

[Bib.40] www.eu-datagrid.org

[Bib.41] S. Andreozzi, at all *GridICE:Requirements, Architecture and Experience of Monitoring Tool for Grid Systems*, Proc. CHEP 2006 Mumbai.

[Bib.42] Andreozzi, De Bortoli, Fantinel, Ghiselli, Rubini, Tortone, Vistoli, *GridICE: A Monitoring Service for the Grid*, 3rd Cracow Grid Workshop, 2003.

[Bib.43] ZaniKolas, Sakellarious, *A Taxonomy of Grid Monitoring Systems*, Future Generation Computer Systems, page 163–188, 2004

[Bib.44] M. Price, *The LHC Project*, Nuclear Instruments and Methods in Physics Research Section A., Vol 478, Issues 1–2, pag. 46–61 2002

[Bib.45] Hosckek, Jean-Martinez, Samar, H. Stockinger, K. Stockinger, *Data Management in an International Data Grid Project*, Lecture Notes in Computer Science, Vol 1971/2000, 2000.

[Bib.46] http://www.euchinagrid.org/

[Bib.47] http://www.na.infn.it/

[Bib.48] http://lcg.web.cern.ch/LCG/

[Bib.49] Andreozzi, Fattibene, De Bortoli at all: *Flexible Notification Service for GRID Monitoring Events* Chep 06

[Bib.50] Xuechai Zhang Freschl, Schopf: *A performance study of monitoring and information services for distributed systems*, High performance Distributed Computing, Proceedings, 12th International Symposium, 2003

[Bib.51] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman, *An effi cient multicast protocol for content-based publishsubscribe systems.* 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99), pages 262–272, Austin, Texas,1999.

[Bib.52] Fabret, Hacobsen, Lirbat, Pereira, *Filtering algorithms and implementation for very fast publish/subscribe systems*, In ACM SIGMOD 2001, pages 115–126, Santa Barbara, California, 2001.

[Bib.53] Andreozzi, G. L. Rubini, Fantinel, Legnaro, De Bortoli, G. Tortone, *A Multidimensional Approach to the Analysis of Grid Monitoring Data*, proceedings CHEP 2004

[Bib.54] A. Rowstron, P. Druschel, *Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems* Proc. Conf. Distributed Systems Platforms (Middleware)-ACM Press, 2002.

[Bib.55] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduzione agli algoritmi*. Jackson libri, 1999.

[Bib.56] Carzaniga, Rosenblum, Wolf, Achieving *scalability and expressiveness in an internetscale event notification service*, Proc. 19th ACM Symposium on Pinciples of Distributed Computing (PODC 2000), Jul 2000.

[Bib.57] R. Chand, P.A. Felber. *A Scalable Protocol for Content-Based Routing in Overlay Networks*, CiteSeer 2003.

[Bib.58] http://infnforge.cnaf.infn.it/gridice/index.php/NotGuide/HomePage

[Bib.59] http://gridice4.cnaf.infn.it:50080/gridice/help/SE-site-help.html]