| | |
|---|---|
| **Project no.:** | **IST-FP6-STREP - 027513** |
| **Project full title:** | **Critical Utility InfrastructurAL Resilience** |
| **Project Acronym:** | **CRUTIAL** |
| **Start date of the project:** | **01/01/2006 Duration: 36 months** |
| **Deliverable no.:** | **D4** |
| **Title of the deliverable:** | **Preliminary Architecture Specification** |

**Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)**

| | |
|---|---|
| **Contractual Date of Delivery to the CEC:** | **17/01/2007** |
| **Actual Date of Delivery to the CEC:** | **17/01/2007** |
| **Organisation name of lead contractor for this deliverable** | **FCUL** |
| **Editor(s): Nuno Neves[2]; Paulo Veríssimo[2]** | |
| **Author(s): Anas Abou El Kalan[4]; Amine Baina[4]; Hakem Beitollahi[5]; Alysson Bessani[2]; Andrea Bondavalli[3]: Miguel Correia[2]; Alessandro Daidone[3]; Geert Deconinck[5]; Yves Deswarte[4]; Fabrizio Grandoni[3]; Nuno Neves[2]; Tom Rigole[5]; Paulo Sousa[2]; Paulo Veríssimo[2]** | |
| **Participant(s): (1) CESI; (2) FCUL; (3) CNR-ISTI; (4) LAAS-CNRS; (5) KUL; (6) CNIT** | |
| **Work package contributing to the deliverable:** | **WP4** |
| **Nature:** | **R** |
| **Dissemination level:** | **PU** |
| **Version:** | **004** |
| **Total number of pages:** | **106** |

**Abstract**

The largely computerised nature of critical infrastructures on the one hand, and the pervasive interconnection of systems all over the world, on the other hand, have generated one of the most fascinating current problems of computer science and control engineering: how to achieve resilience of critical information infrastructures, in particular in the electrical sector.

This problem is concerned with ensuring acceptable levels of service and, in last resort, the integrity of systems themselves, when faced with threats of several kinds. In this document we are concerned with threats against computers and control computers, not the physical infrastructures themselves. Moreover, we focus on systems with great socio-economic value, such as utility systems like electrical, gas or water, or telecommunication systems and computer networks like the Internet.

Although there is an increase in the concern for using security best practices in these systems, we believe that the problem is not completely understood, and can not be solved with classical methods. Its complexity is mainly due to the hybrid composition of those infrastructures: operational network, called generically SCADA, devoted to the physical processes; corporate intranet, where usual departmental services and clients reside; Internet, through which intranet users get to other intranets and/or the outside world, but to which, and often unwittingly, the SCADA network is sometimes connected to.

In consequence, in scientific terms, our problem can be formulated as follows: *The computer-related operation of a critical utility infrastructure is a distributed systems problem including interconnected SCADA/embedded networks, corporate intranets, and Internet/PSTN access subsystems. This distributed systems problem is hard, since it simultaneously includes facets of real-time, fault tolerance, and security*.

In this document, we describe the preliminary architecture specification for a CRUTIAL infrastructure. The document is divided in three main parts. The first part introduces the main characteristics of the system model, which includes the fault, synchrony and topological models. Then, the second part, presents the major architectural options and components. The last part, describes the building blocks of the CRUTIAL middleware.

**Keyword list: Electrical Critical Infrastructures; Architecture; Middleware**

## DOCUMENT HISTORY

| Date | Version | Status | Comments |
|---|---|---|---|
| 1 Nov | 001 | Int | Preliminary version with contents from CRITIS |
| 1 Dec | - | - | Partner contributions |
| 15 Dec | 002 | Int | Integration of material from partners |
| 22 Dec | - | - | Comments |
| 5 Jan | 003 | Int | Second draft |
| 10 Jan | - | - | Comments |
| 17 Jan | 004 | Apr | Final version |
|  |  |  |  |

# Table of Contents

# 1 INTRODUCTION

The largely computerised nature of critical infrastructures on the one hand, and the pervasive interconnection of systems all over the world, on the other hand, have generated one of the most fascinating current problems of computer science and control engineering: *how to achieve resilience of critical information infrastructures, in particular in the electrical sector*.

This problem is concerned with ensuring acceptable levels of service and, in last resort, the integrity of systems themselves, when faced with threats of several kinds. In this document we are concerned with threats against computers and control computers, not the physical infrastructures themselves. These threats range from accidental events like natural faults or wrong manoeuvres [Madani & Novosel 2005][van Eeten et al. 2006], to attacks by hackers or terrorists [Cieslewicz 2004][Li et al. 2005][Luiijf & Klaver 2004][Pollet 2002][Wilson 2006]. The problem affects systems with great socio-economic value, such as utility systems like electrical, gas or water, or telecommunication systems and computer networks like the Internet. In consequence, the high degree of interconnection is causing great concern, given the level of exposure of very high value systems and components to attacks that can be perpetrated in an anonymous and remote way.

Although there is an increase in the concern for using security best practices in these systems [Byres et al. 2005][US_Energy 2002], we believe that the problem is not completely understood, and can not be solved with classical methods. Its complexity is mainly due to *the hybrid composition of those infrastructures*:

- The operational network, called generically SCADA (Supervisory Control and Data Acquisition)[1], composed of the computer systems that yield the operational ability to supervise, acquire data from, and control the physical processes. In fact, to the global computer system, SCADA computer systems (e.g., controllers) “are" the controlled processes (e.g., power generators), since by acting on the former, for example, through a network message, one changes the state of the latter.
- The corporate intranet, where usual departmental services (e.g., web, email, databases) and clients reside, and also the engineering and technical staff, who access the SCADA part through ad-hoc interconnections[2].
- The Internet, through which intranet users get to other intranets and/or the outside world, but to which, and often unwittingly, the SCADA network is sometimes connected to.

Besides the complexity due to this hybrid composition, this mixture has given an unexpected *inter-disciplinary nature* to the problem: SCADA systems are real-time systems, with some reliability and fault tolerance concerns, but they were classically not designed to be widely distributed or remotely accessed, let alone open to other more asynchronous and less trusted subsystems. Likewise, they were not designed with security in mind. In consequence, in scientific terms, our problem can be formulated as follows:

The computer-related operation of a critical utility infrastructure is a distributed systems problem including interconnected SCADA/embedded networks, corporate intranets, and Internet/PSTN[3] access subsystems. This distributed systems problem is hard, since it simultaneously includes facets of real-time, fault tolerance, and security.

[1] Or PCS (Process Control System).

[2] In some companies there is a (healthy) reluctance against interconnecting SCADA networks and the corporate network or the Internet. However, in practice this interconnection is a reality in many companies all over the world. We believe this is indeed the situation in most companies and this is the case we are interested in this document.

[3] Public Switched Telephone Network.

In this document, we describe the preliminary architecture specification for a CRUTIAL infrastructure. The document is divided in three main parts. The first part introduces the main characteristics of the system model, which includes the fault, synchrony and topological models. Then, the second part, presents the major architectural options and components. The last part, describes the building blocks of the CRUTIAL middleware.

# 2 SYSTEM MODEL

Before we proceed, let us bring some further insight on the problem of critical infrastructures:

- Critical Information Infrastructures (CII) feature a lot of legacy subsystems and non-computer-standard components (controllers, sensors, actuators, etc.).
- Conventional security and protection techniques, when directly applied to CII controlling devices, sometimes stand in the way of their effective operation.

These two facts will not change, at least for a long time, so they should be considered as additional research challenges. Despite security and dependability concerns with those individual components being a necessity, we believe that the crucial problem is with the forest, not the trees. That is, the problem of critical information infrastructure insecurity is mostly created by the informatics nature of many current infrastructures, and by the generic and non-structured network interconnection of CIIs, which bring several facets of exposure, from internal unprotected wireline or wireless links, to interconnections of SCADA and corporate intranets to the Internet and PSTN. This situation is conspicuous in several of the attacks reported against CIIs. For instance, the attack of the *Slammer* worm against the Davis-Besse nuclear power plant (US) was due both to this combination of a computerised CII with non-structured network interconnections and lack of protection [Geer 2006]. Although the network was protected by a firewall, the worm entered through a contractor's computer connected to the CII using a telephone line.

The problems that may result from this exposure to computer-borne threats range from wrong manoeuvring to malicious actions coming from terminals located outside, somewhere in the Internet. The potential targets of these actions are computer control units, embedded components and systems, that is, devices connected to operational hardware (e.g., electrical power generators and power protections, water pumps and filters, dam gates, etc.) or to telecom hardware (core routers, base stations, etc.). The failure perspectives go from unavailability of services supposed to operate 24X7, to physical damage to infrastructures. In the electrical power provision these situations have already been witnessed [Dondossola et al. 2006]: among the blackouts that occurred in several countries during the summer of 2003, the analysis report [US-Canada 2003] of the North American one highlighted the failure of various information systems as having thwarted the utility workers' ability to contain the blackout before it cascaded out of control, leading to an escalating failure.

Whilst it seems non-controversial that such a status quo brings a certain level of threat, we know of no work that has tried to equate the problem by defining a reference model of a *critical information infrastructure distributed systems architecture*, providing the necessary global resilience against abnormal situations.

We believe that evaluation work based on such a model will let us learn about activity patterns of interdependencies, which will reveal the potential for far more damaging fault/failure scenarios than those that have been anticipated up to now. Moreover, such a model will be highly constructive, for it will form a structured framework for: conceiving the right balance between prevention and removal of vulnerabilities and attacks, and tolerance of remaining potential intrusions and designed-in faults.

What can be done at architectural level to achieve resilient operation? Note that the crux of the problem lies with the fact that access to operational networks, such as remote SCADA manoeuvring, ended up entangled with access to corporate intranets and to public Internet, without there being computational and resilience models that *represent* this situation, unlike what exists in simpler, more homogeneous settings, e.g. classical web-based server infrastructures on Internet. Our point is that interference and threats start at the level of the macroscopic information flows between these subsystems, and can in consequence be stopped there. This should not prevent the study of techniques at the controller level, but in this paper we will not focus on this latter issue.

Now, given the simultaneous need for real-time, security and fault tolerance, this problem is hard vis-a-vis existing paradigms. For example, many classical distributed systems paradigms handle each of those facets separately, and just solve part of the problem. A unifying approach has gained impressive momentum currently: *intrusion tolerance* [Verissimo et al. 2003]. In short, instead of trying to prevent every single intrusion or fault, they are allowed, but tolerated: systems remain to some extent faulty and/or vulnerable, attacks on components can happen and some will be successful, but the system has the means to trigger automatic mechanisms that prevent faults or intrusions from generating a system failure.

Our approach is thus equated along the following propositions:

**PROPOSITION 1:** Classical security and/or safety techniques alone will not solve the problem: they are largely based on prevention, intrusion detection and ad-hoc recovery or ultimately disconnection.

There is a recent and positive trend to make SCADA systems and CIIs at large more secure [US_Energy 2002][Byres et al. 2005][Li et al. 2005][Stamp et al. 2003][Stouffer et al. 2006]. However, classic engineering remedies place real-time and embedded (RTE) systems at most at the current level of commercial systems' security and dependability, which is known to be insufficient [Cieslewicz 2004][Gordon et al. 2006][Turner et al. 2005]: systems constantly suffer attacks, intrusions, some of them massive (worms); most defences are dedicated to generic non-targeted attacks; attacks degrade business but only do virtual damage, unlike RTE systems where there is a risk of great social impact and even physical damage. On the other hand, some current IT security techniques can negatively affect RTE system operation, w.r.t. availability and timeliness. For example, if security is based on disconnection, significant performance degradation, or even defensive restrictions can prevent the actuation or monitoring of the infrastructure.

**PROPOSITION 2:** Any solution, to be effective, has to involve automatic control of macroscopic command and information flows, occurring essentially between the physical or virtual LANs[4] composing the critical information infrastructure architecture, with the purpose of securing appropriate system-level properties.

We believe that a key to the solution lies with controlling the command and information flow at macroscopic level (organisation-level). We are talking about an architectural model, a set

[4] Local Area Networks.

of architectural devices, and key algorithms, capable of achieving the above-mentioned control of the command and information flow.

The devices and algorithms should be capable of securing a set of system-level properties characterising whatever is meant by correct and resilient behaviour.

**PROPOSITION 3:** We lack a reference architecture of “modern critical information infrastructure" considering different interconnection realms and different kinds of risk, throughout the physical and the information subsystems of a CII.

We must consider the physical or virtual LANs composing the operational SCADA/embedded networks, the corporate intranets, and the Internet/PSTN access networks, as different first order citizens of the architecture. Likewise, the notion that risk factors may vary and be difficult to perceive accurately, brings the need to reconcile uncertainty with predictability in architecture and algorithms.

## 2.1 Fault Model

Being the CRUTIAL infrastructure composed by an information networked system and the electric power system, it is necessary to take into account not only problems involving each infrastructure in isolation, but also problems originated from one infrastructure that could negatively affect the other (e.g., the ICT infrastructure needs power coming from the power system, the power system needs ICT communication in order to control the system itself, …). The entire system at all the abstraction levels has to be partitioned in order to identify FRUs (Field Replaceable Units).

In the following subsection the above domains are presented.

### 2.1.1 Power System Failures

The term electrical system covers all infrastructures, the technologies and, generally speaking, all the devices necessary to produce and to transport the electric power towards the final users. Electric power is obtained transforming several kinds of energy available in nature by means of machines called generators, situated inside power plants. The energy produced by the generators is then adapted by components called transformers, to be conveyed with minimal dispersion, to the different types of end users (civil, industrial, military, etc) [Crutial_D2 2007].

The infrastructure used to transport the electric power from the production plants to the final customers is called *power grid*. The main elements that constitute the power grid are the following:

- *Transmission lines*: they are components which physically connect the substations with the power plant and the final users
- *Substations*: they are structured components in which the electric power is transformed and split over several lines. In the substations there are transformers and several kinds of connection components (e.g., *bus-bars*, *bays*, *switches* and *breakers*)

A disturbance or contingency is the unexpected failure or outage of a system component, such a generator, transmission line, circuit breaker, switch, or other electrical component. A contingency also may include multiple components, which are related to situations leading to simultaneous component outages [UCTE 2006].

The main causes of a system component failure could be summarized in [Fink and Carlsen 1978]:

- Internal electrical or mechanical faults
- Malfunction of protective or control device
- External events such a lightning, floods, terrorism

For the failures originated by internal faults of a given component class it is common to use reliability models expressing the failure probability in relation with the fault rate, the age and the maintenance interventions of the component. Fault rate data are normally available for elementary components only and it is necessary to correlate such data with the failure of macro-components normally modelled by power system simulators. Also the failure probability may depend on the current operation conditions, as in the case of power lines.

For the failures caused by external events it is necessary to correlate reliable historical data on power contingencies (e.g. breaker trips due to protection interventions) with environmental (geographical and meteorological) events.

A deep study on some of these aspects is currently ongoing within the RdS programme [RdS 2006] in the context of the development of risk assessment tools based on probabilistic approaches.

The main electrical contingencies (or critical electrical phenomenon) could be summarized in:

- Unexpected increasing of demand
- Tripping of great generation groups
- Transient or permanent overloads of lines or transformers
- Instability of voltage
- Collapse of voltage
- Important cascading, line tripping with consequent grid separation.

The extreme power system failure is of course the generalised black-out of a whole power system.

The North American Electric Reliability Council (NERC) has a documented list summarizing the disturbances and major blackouts of the North American power transmission system since 1984. Several statistical analyses of this or similar data exist in the literature. The main observation is that the probability distribution of the blackout data does not decrease exponentially with the size of the blackout, but rather has a heavy-tail distribution that is fairly close to a power law. The consequence of the power law is that the risk of large blackouts is the same or even greater than the risk of small blackouts [Crutial_D2 2007].

### 2.1.2 Information System Failures

The wide area interconnected power system needs quite sophisticated controls that are implemented mostly by a variety of information subsystem interconnected with the power system. Power controls are typically arranged in a hierarchical structure.

- Primary control: it concerns the local control of generators, implemented by special-purpose electronics and programmable controllers. The primary control could be performed without the need of communication with the rest of the power system, just seeking to maintain equilibrium on local electrical and mechanical parameters.
- Secondary control: it aims to regulate the power production / distribution inside a given area, according to some strategy; each strategy dynamically sets generation local parameters to be commanded to the primary control. The secondary control needs relatively fast communications among area's substations, and it is vital for the power system.
- Tertiary control: it aims to optimize energy losses and marketing aspects, so it does need communications. The tertiary control is not vital for the power system management; in fact, if it were inactive, the power grid would be still operative, albeit in a sub-optimal condition.

The information system malfunctions could be classified as follows:

- *Hardware faults*: they could occur in different physical components leading to the following problems:
    - in the devices directly related to the primary control of the power system (SCADA devices); these faults can create problems as "undue tripping"[5] or "missing action";
    - at communication level, leading to lost packets (packet lost on a dying link) or late packets (some packets could be routed on too long paths);
    - in the computers performing the secondary (and the tertiary) control or supporting corporate networks; those faults can lead to the crash of a part of the information system or to erroneous command sequences (value faults).
- *Communication related faults*: omitted messages, burst losses (due to intermittent physical disconnections), late messages, unexpected messages, wrong values, Byzantine faults, network partitioning;

- *Software related faults in the information nodes*: while most faults at this level can manifest as application/system crashes, the possibility of a less favourable behaviour is not ruled out (e.g. issuing wrong and/or inconsistent messages on the verge of crashing). Examples of software faults are application errors, O.S. related errors, resources exhaustion, late services...

- *Malicious faults* [Powell & Stroud 2003]:
    - attack: malicious interaction fault through which an attacker aims to deliberately violate one or more security properties (an attack is an intrusion attempt);
    - vulnerability: a security hole left out during the development of the system or opened during operation;
    - intrusion: a malicious, externally-induced fault resulting from an attack that has been successful in exploiting a vulnerability.

  An example of malicious attack is the denial of service; examples of intrusions are Trojan horses, worms, viruses…

  A security failure at one level of decomposition of the system may be interpreted as an intrusion at the next upper level (e.g. the failure of an authentication and authorization mechanism to prevent system penetration by a malicious user is an intrusion as seen from the containing system).

[5] Tripping is the action of the opening of a breaker.

The concept of "Intrusion tolerance" can be introduced (how to provide correct service in the presence of intrusions). Intrusion Containment Region (ICRs) could be introduced by analogy with the notion of Fault Containment Regions[6] (FCR) in order to guarantee certain security properties despite of the fact that some components could be compromised.

## 2.2 Synchrony Model

The synchrony model identifies the main assumptions on the duration of system events. Research in distributed systems has traditionally been based on two models, the fully asynchronous and the fully synchronous models. The asynchronous model is characterized by an absence of timeliness specifications, i.e., no dependencies exist on time. In contrast, a synchrony model imposes known limits on the execution time of every operation of the system, being these operations local or distributed (e.g., transmission of a message). In between these two extreme models, there are a number of other models, such as partial synchrony models and the wormhole.

On a CRUTIAL infrastructure, which encompasses various kinds of networked systems, from SCADA to the ICT infrastructure, one expects that different kinds of models apply. On the rest of this section, we will review the main synchrony models that are relevant to the distinct parts of the architecture.

### 2.2.1 Asynchronous Model

The asynchronous time model is the weakest synchrony model because no assumptions are made about the relative speed of processes, and no bounds exist for the transmission of messages through the network. A distributed system based on this model is characterized by the following properties[7]:

Pa1. Unbounded or unknown processing delays

Pa2. Unbounded or unknown message delivery delays

Pa3. Unbounded or unknown rate of drift of local clocks

Pa4. Unbounded or unknown difference of local clocks

The asynchronous model is particularly interesting for the parts of the infrastructure where malicious attacks can occur (e.g., Internet or ICT infrastructure). Attacks can for instance be directed to create a violation on some assumption about an upper bound for an operation (e.g., through simple denial of service attacks on nodes or networks). Protocols developed under this model are immune to these attacks because their correctness cannot be compromised due to these artificial delays.

On the other hand, due to their time-free nature, asynchronous models cannot be used to solve timed problems. For example, they cannot address Quality of Service (QoS) specifications, which are of increasing importance in the measure of the quality of transactional systems in open networks such as the Internet. Moreover, they cannot be employed on control systems where a bound needs to be enforced betweens the instants,

[6] A fault containment region (FCR) can be defined as a set of components that is considered to fail (i) as an atomic unit, and (ii) in a statistically independent way with respect to other FCRs.

[7] One should note that a clock on a time-free model is nothing more than a sequence counter, and for this reason talking about clock synchronization does not make much sense.

the operator issues a command and the power generator controllers perform the specified action.

In addition, asynchronous models preclude the deterministic solution of several relevant problems, such as consensus, total order multicast, or atomic commitment [Fischer et al. 1985]. In this scenario, only probabilistic (or randomized) solutions can be employed. Besides, the only way asynchronous models can reason in terms of causality between events is in a logical way, and this is insufficient if hidden communication channels exist between participants.

### 2.2.2 Synchronous Model

A synchronous time model allows the specification of timeliness requirements. This model is characterized by having known bounds for processing and message delivery delays, and for the drift rate of the local clocks. In consequence, this model solves timed problem specifications, which is a precondition for a subset of the information systems of the electrical infrastructure. For example, the primary power controls have strict needs in terms of time for certain actions carried out by the programmable controllers. Synchronous systems are characterized by the following properties:

Ps1. There exists a known bound for processing delays

Ps2. There exists a known bound for message delivery delays

Ps3. There exists a known bound for the rate of drift of local clocks

Ps4. There exists a known bound for the difference among local clocks

It is easy to see that synchronous models are susceptible to time-based attacks, since they make strong assumptions about the instants when things are supposed to happen and/or the duration of the execution of the system events. Synchronous models are fragile in terms of the coverage of such timeliness specifications. Timing bounds are assumed to be valid for correct processes, and conversely their validity must not be compromised by incorrect processes. Both are difficult to achieve in practice. For example, protocol safety can be compromised if messages do not arrive by a certain time due to instabilities in the communications. Likewise, reading the actual global time from a clock can cause dangerous dependencies, if clocks can be manipulated by an adversary or fail in a Byzantine way [Gong 1992]. For this reason, most recent research on systems where malicious faults can occur does not use this type of models.

This model, however, does have some advantages. For instance it allows the solution of all hard distributed problems (e.g., consensus, atomic broadcast, clock synchronization). Moreover, many of the emerging power grid applications have interactivity or mission-criticality requirements. That is, service must be provided on time, either because of cost optimization limitations (e.g., adequate the power production to current user needs), or because of dependability constraints (e.g., command-and-control applications for emergency networks).

### 2.2.3 Partially-Synchronous Model

The FLP impossibility result sets the stage for the difficulties to be encountered when wanting to solve certain problems in essentially time-free systems: Fischer, Lynch and Paterson showed that any deterministic consensus protocol for asynchronous systems has the possibility of non termination if a single process is allowed to crash [Fischer et al. 1985].

The result was then extended by equivalence to other important distributed systems primitives, such as total order broadcast [Chandra & Toueg 1996].

Several authors have tried to overcome these difficulties in a number of ways, generally around these two important but very specific problems (see also [Keidar & Rajsbaum 2001], [Raynal 2005] for surveys). In the past, basically two solutions have been employed to circumvent the FLP result. The first resorts to the use of randomisation techniques [Ben-Or 1983][Rabin 1983], and the second to extending the basic asynchronous model with some time related assumptions. These assumptions can be made explicitly [Dolev et al. 1987][Dwork et al. 1988], or they can be encapsulated in some construct such as an unreliable failure detector [Chandra & Toueg 1996].

Restrictions to the asynchrony of systems have been addressed in earlier studies [Dolev et al. 1987][Dwork et al. 1988], but partially synchronous models have deservedly received greater attention more recently. They yield better results, essentially for three reasons: (i) they allow timeliness specifications; (ii) they admit failure of those specifications; (iii) they provide timing failure detection.

Cristian & Fetzer devised the timed-asynchronous model, where the system alternates between synchronous and asynchronous behaviour [Christian & Fetzer 1998], making progress when the system has just enough synchronism to make decisions such as 'detection of timing failures'. Another partially synchronous model is the quasi-synchronous model that was proposed by [Verissimo & Almeida 1995]. Here it is assumed that bounds exist for the system properties (process speed, message transmission delay, clock drift rate, …), but some of the chosen values have a non-null probability of being incorrect.

Chandra & Toueg proposed the failure detectors, a very elegant way to structure consensus-related algorithmic, but in a very similar manner they gave a hierarchy of such detectors, requiring the system to be or become synchronous enough, and for long enough periods, in order to solve consensus [Chandra & Toueg 1996].

### 2.2.4 Hybrid Models

In [Verissimo & Casimiro 2002] the following observation was made: *synchronism is not an invariant property of systems*. On the one hand, it was meant that the degree of synchronism varies in the time dimension: during the timeline of their execution, systems become faster or slower, actions have greater or smaller bounds. The works on partial synchrony have indeed relied on this variation as a problem solver. On the other hand, it was meant that it varies with the part of the system being considered, that is, in the space dimension: some components are more predictable and/or faster than others, actions performed in or amongst the former have better defined and/or smaller bounds. This was the innovative perspective at that time, and the insight that led to the Wormholes hybrid distributed system model.

Both dimensions (time or space) are interesting, and they allow a system designer to think about algorithms in an abstract way. However, there is remarkable difference between the two dimensions - under the time dimension one *expects* the system to become adequately synchronous, whereas by exploring the space dimension i.e., acting on the system structure, one *makes* the necessary synchronism happen. That is, one can make some parts of the system exhibit a well-defined (and perpetual if desired) time-domain behaviour, regardless of the asynchronism of the rest of the system.

All the works on partial-synchrony considered the eventual evolution of the system through periods of sufficient synchrony, during its execution: they only explored the time dimension. In essence, this explains why we want to explore in CRUTIAL the hybrid distributed systems models, where different loci of the system have different properties and can rely on different sets of assumptions (e.g., faults, synchronism). These models allow us to take the best from both dimensions, both in theoretical and practical terms.

#### 2.2.4.1 Wormhole-based Hybrid Models vs. Homogeneous Models

Wormhole-based hybrid distributed systems models are:

- *Expressive models w.r.t. reality* - Real systems are partially synchronous in the time dimension. But further to that, they generally have components with different degrees of synchronism, i.e., in the space dimension: different bounds or absence thereof, different speeds, different tightness and steadiness (metrics of synchronism, see [Verissimo & Rodrigues 2001]). Homogeneous models simply cannot take advantage from this, being confined to use the worst-case values or bounds (e.g., of the least synchronous component), which ultimately— and safely— mean asynchrony.
- *Sound theoretical basis for crystal-clear proofs of correctness* - Why were some problems found in earlier-generation asynchronous algorithms [Chandra et al. 1996] [Anceaume et al. 1995]? One of the reasons was because timing assumptions were made for the system that were not in agreement with the model. Artificially restricting such assumptions to 'parts' of an asynchronous system does not improve the situation much if it follows a homogeneous model: we may fall into the same kind of contradictions. However, by using a hybrid model, the heterogeneous properties of the different loci of the system (the space dimension...) are by nature represented, and we are in consequence forced to explicitly make correctness assertions about each of these loci, and about the interfaces to one another.
- *Naturally supported by hybrid architectures* - Sisters to hybrid distributed systems models, hybrid architectures accommodate the existence of actual components or subsystems possessing different properties than the rest of the system. Hybrid models and architectures provide feasibility conditions for powerful abstractions which are to a large extent unimplementable on canonical (homogeneous) asynchronous models: failure detectors; ad-hoc synchronous channels; timely execution triggers (a.k.a. watchdogs) or timely executed actions (periodic or event-triggered). Hybrid models and architectures may drastically increase the usefulness and applicability of all these abstractions.
- *Enablers of concepts for building totally new algorithms* - A powerful yet simple concept behind the first experiments with hybrid models was: use the weakest possible model for the generic system; imagine that a "toolbox" of simple but stronger low-level services is available, locally accessible to processes (e.g., timely execution triggers; timely executed actions; trusted store); these local services can be distributed via alternative channels, to obtain further strength (e.g. synchronous channels; trusted global time; trusted binary agreement); devise algorithms which, by working between the two space-time realms, the generic and the enhanced subsystem containing the "toolbox", achieve new properties (e.g., making an asynchronous process enjoy timely execution).

#### 2.2.4.2 Properties of a Wormhole-base Model

A hybrid distributed system model features several subsystems following different sets of assumptions, e.g. about synchrony or faults. In theory, nothing prevents a hybrid model where, for example, several synchronous subsystems coexist with several asynchronous ones. However, note that the instance of such model should meet the best practice of using the simplest possible model with the weakest possible assumptions – typically, one would have a weak main or 'payload' subsystem, and a few small, simple wormholes.

For the sake of simplicity and without loss of generality, we assume a bi-modal system, with one payload system, and one wormhole subsystem, more complex systems can be recursively defined:

- There is a payload system *Sp* where algorithms and applications normally execute, composed of *Np* payload processes *pi* that communicate via message passing, through payload channels.
- *Sp* follows a set of fault and synchrony assumptions *Hp* (normally weak, such as processing and communication being asynchronous, and faulty behaviour Byzantine).
- There is a wormhole subsystem *Sw* composed of *Nw* wormhole processes *wi*. Wormhole processes may or not communicate amongst themselves, in which case they do so via message passing, through wormhole channels.
- *Sw* follows a set of fault and synchrony assumptions *Hw* normally stronger than the payload (such as processing and communication being synchronous, and faulty behaviour crash[8]).
- The only way for payload processes to communicate with wormhole processes is through wormhole gateways, *WG*, with well-defined interfaces. The specific type of interface is not part of the model.
- Likewise, for payload processes the properties offered by any wormhole are defined and enjoyed at a wormhole gateway. The payload processes do not have to know how wormholes are implemented, and vice-versa.
- The relative number of payload and wormhole processes is not part of the model. In fact, maybe not all payload processes access wormholes in certain algorithms, or maybe more than one payload process can access a same wormhole.

In practical terms, a wormhole is a privileged artefact to be used only when needed, and supposedly implements functionality hard to achieve on the payload system, which in turn should run most of the computing and communications activity.

Note that the payload and the wormholes follow different sets of assumptions, but there is no preassumption about what these sets are. For example, the payload may be asynchronous and/or have Byzantine failures, but it may also have some synchrony for a start (imagine wormhole-aware real-time systems...). Likewise, a wormhole may be synchronous, partially synchronous, etc. as fits the needs of the problems to solve.

So, in summary, the key innovative characteristic of the Wormholes model consists in making some stronger properties (e.g., synchrony) happen in a well-defined and safe way, whilst preserving the canonical model's weak abstractions (e.g., asynchrony). Note that whilst the most fascinating and powerful incarnation of a wormhole would be distributed (through alternative channels with enough synchrony), the simpler versions, local wormholes, still provide very useful support (e.g., local security and/or timeliness functions).

Figure 1 gives a picture of the model just presented, in a bi-modal incarnation. Figure 1(a) suggests a model with local wormholes, whereas Figure 1(b) depicts a distributed counterpart. The local wormholes case is the special case of this model where wormhole processes do not communicate directly with one another.

[8] This is the extreme, for the sake of example, but we hope to have left clear that wormholes can assume any weaker synchrony or fault model.

This model ensures a clear separation of concerns between the properties offered by wormholes and their construction, and between the executions on either side of a wormhole gateway. For instance, consider a system with an asynchronous payload and timely execution wormholes, which guarantee that a function *f* invoked at their interface is always executed in a known bounded time. It is easy to reason about the invocation of *f* on a wormhole *wu*, by a process *pk*. Using whatever invocation method stipulated, the timing of *pk* getting to invoke *f* at the interface concerns the payload system realm. The timing of *wu* getting to execute *f* since invoked concerns the wormhole subsystem realm. Likewise if a possible result is expected to be returned by *f*: the time from invocation to return at the interface is what gets bounded in the wormhole realm, whereas the timing for *pk* to use the result pertains to the payload realm. To give an intuition about reality, in this example one might imagine each *pi* co-located with a local wormhole *wi* implemented by some sort of real-time micro-kernel, accessed by *pi* as a run-time system call. Although it seems counterintuitive, with slightly more sophistication in the exemplified interface, asynchronous processes can indeed take advantage from synchronous executions. The reader is referred to [Neves et al. 2005][Verissimo et al. 2000], where such techniques are described.

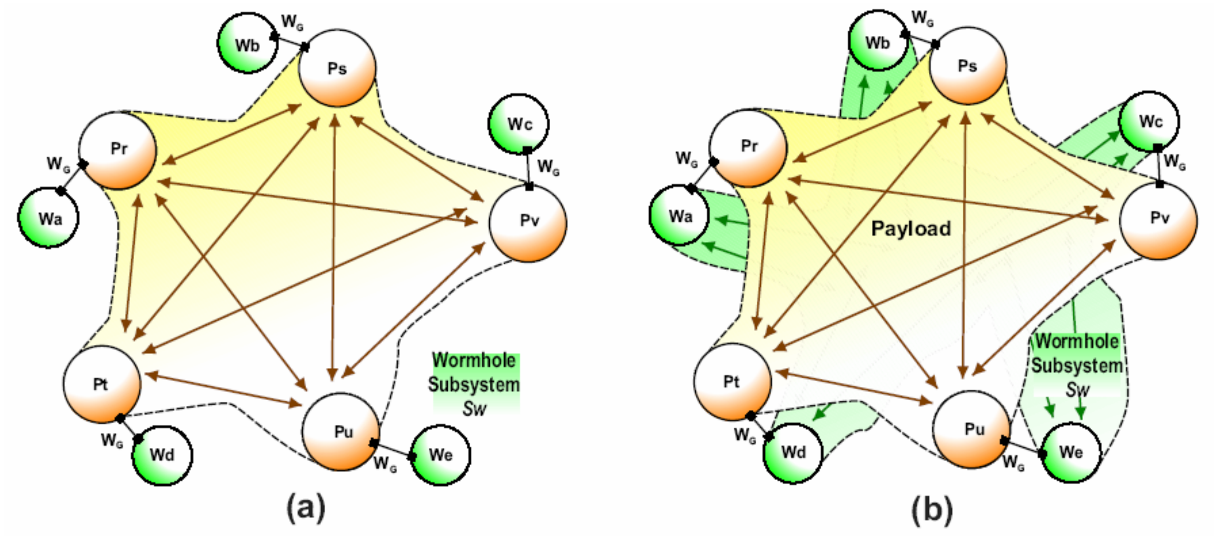

**Figure 1: The Wormhole model: (a) Local Wormholes; (b) Distributed wormholes.**

### 2.2.4.3 Architectural Hybridisation

Recapitulating the observations made above, hybrid modelling of distributed systems is the path to achieving incrementally stronger behaviour taking the best of two worlds: retaining the canonical and useful weakest assumptions paradigm of asynchronism; achieving synchronism in a predictable manner.

Architectural hybridisation was proposed as a new paradigm to architect modular systems, based on a few simple principles:

- Systems may have realms with different non-functional properties, such as synchronism, faulty behaviour, quality-of-service, etc.
- The properties of each realm are obtained by construction of the subsystem(s) therein.

- These subsystems have well-defined encapsulation and interfaces through which the former properties manifest themselves.

As to the construction, architectural hybridisation is an enabler of the construction of realistic hybrid distributed systems [Verissimo 2003]. In fact, it is quite straightforward to build architecturally-hybrid systems. Some earlier systems already exhibited flavours of architectural hybridisation [Powell et al. 1988][Verissimo et al. 1997]. More recently, a few experimental wormhole systems have been built supported by architectural hybridisation, and the feasibility of these implementations, both in the time and security facets, is amply discussed in [Casimiro et al. 2000][Correia et al. 2002][Verissimo 2003]. A timeliness and security distributed wormhole implementation is downloadable from the web[9], for experimental work with distributed algorithms on hybrid systems.

As to the usefulness of wormholes, the reader is referred to [Verissimo 2003] for an overview of the multiple potential uses of wormholes. In one of our experiments, we have prototyped a specific kind of wormhole subsystem called Timely Computing Base (TCB) that allows achieving timely actions in systems that can be asynchronous. At the TCB distributed wormhole gateway, a set of simple but extremely helpful services are provided: timely execution; duration measurement; timing failure detection. We introduced a technique to interface a synchronous subsystem from a time-free one, making the asynchronous system perform timely (synchronous) actions or detect the failure thereof [Verissimo et al. 2000]. In [Verissimo & Casimiro 2002] we present a formal embodiment of the TCB model and architecture.

The power of a wormhole such as the Timely Computing Base is interesting: immersed in an asynchronous system, it makes feasible the construction of useful abstractions such as perfect failure detectors, triggers such as watchdogs, periodic task dispatchers, or even mere synchronous communication channels [Helary et al. 2000][Fetzer 2003][Castro & Liskov 2002][Friedman et al. 2005][Zhou et al. 2002]. However, it is important to note that a TCB is sufficient to implement a number of interesting paradigms, but it is by no means necessary to implement all of them. The TCB, fully synchronous and distributed, was intended as a proof of concept of wormholes, and not the ultimate wormhole. If it proved, as it did, to be easy to implement such a component— and feasible even in large scale settings [Verissimo 2003]— then doors would be open for weaker and simpler forms of wormholes: subsets of the TCB services, including just local services; raw services such as synchronous bare channels or trusted real-time clocks; partially synchronous variants; trusted variants for security purposes, etc.

In the malicious failure domain, resilience to intrusions (a.k.a. intrusion tolerance) can be drastically augmented by using wormholes. In another experiment, we devised a set of new functions resilient to malicious faults for this new wormhole, calling it Trusted Timely Computing Base [Correia et al. 2002], and showed ways to perform trusted actions in the presence of uncertain attacks and vulnerabilities, such as solving consensus quite efficiently [Correia et al. 2005][Neves et al. 2005].

## 2.3 Topological Model

The topology of the information services that manage the power grid infrastructure is not uniform, since it can be viewed at different levels. Typically, a critical information infrastructure is formed by facilities, like power transformation substations or corporate offices, which are modeled as collections of LANs that are interconnected by a wider-area network, modeled as a WAN. This WAN-of-LANs modeling approach allows for simpler

[9] http://www.navigators.di.fc.ul.pt/software/tcb/index.htm

solutions to hard problems, such as legacy control sub-networks, and interconnection of critical and non-critical traffic.

In the past, previous research in large-scale open distributed systems has taken advantage of this topology awareness in the construction of efficient protocols [Rodrigues & Verissimo 2000]. The basic principle is relatively simple to understand: the mechanisms and protocols are designed to recognize the specificities of the system topology and to take advantage of it, both in terms of functionality and performance. This objective is attained by creating adequate physical topologies inside the organization and by logically reorganizing existing physical topologies when needed (e.g., in the Internet).

Two levels of topology awareness will be exploited in CRUTIAL: inside a node, the site-participant separation divides host-level tasks from individual application tasks; at the network, the WAN-of-LAN concept splits the communication inside local groups of nodes (or sites), which are called *facilities*, from the wide-area communications amongst facilities. The WAN-of-LAN principle can be applied recursively throughout the organization, as is going to be explained below.

### 2.3.1 Site-Participants Separation

The CRUTIAL architecture can support the communication and coordination among the relevant nodes located in the infrastructure. A node can be divided into two parts: the site part connects the node to the network and takes care of inter-node operations, e.g., communication and site failure detection; the participant part uses the services provided by its respective site part to carry out the necessary operations. *Participants* are the generic entities that execute the distributed activities, and can be senders or receivers of information (e.g., to manage control operations of the power stations).

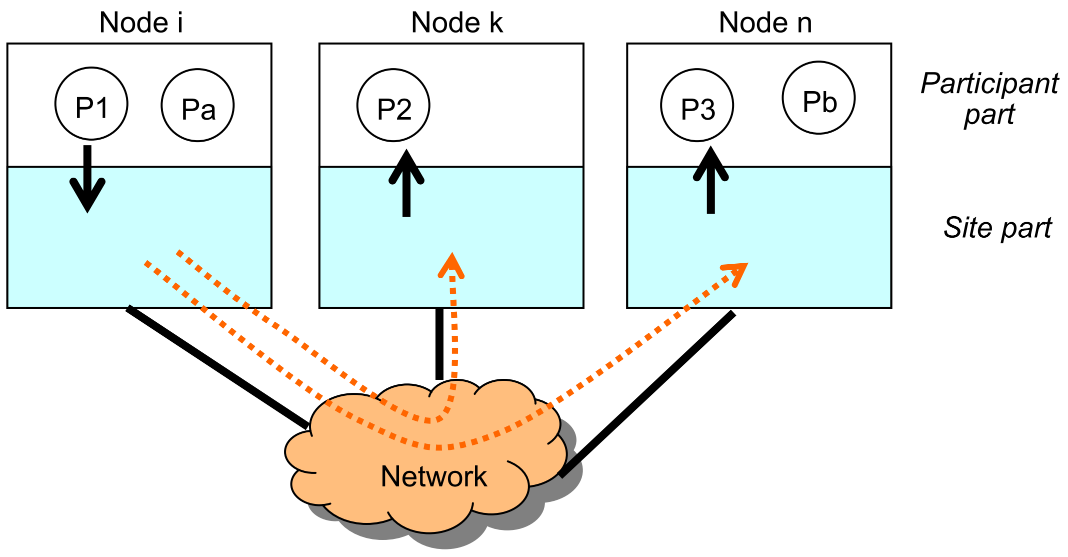

**Figure 2: Site-participant local topology.**

Figure 2 shows a representation of the duality between the site-participant division. The distinction between sites and participants is in favor of a communication subsystem approach for structuring the machine's networking, where the site part performs for instance multiplexing and de-multiplexing operations on the traffic generated by the local participants. Therefore, a site-level protocol server takes care of all decisions regarding the most effective way (e.g., in terms of efficiency) to send or receive data.

### 2.3.2 WAN-of-LANs Separation

Clustering is an interesting technique to cope with large-scale distributed systems, allowing the effective implementation of divide-and-conquer strategies. Whilst it enhances scalability and performance, it also provides hooks for the combined implementation of accidental fault and intrusion tolerant mechanisms (e.g., group communication and access control protocols) and fault-preventive protection strategies (e.g., firewalls). In the context of CRUTIAL, we identify an interesting clustering opportunity – the facility corresponds to a cluster of nodes (or sites) that share common level of access control and administration (e.g., the power generator facility, the administrative office, the supervisory control station facility). These facilities are then interconnected by WAN protocols, creating the abstraction of a WAN-of-LANs.

This structure offers opportunities for making different assumptions regarding the types and levels of threat and degrees of vulnerability of the networks local to a facility versus the global network part. In practice, this does not necessarily mean that intra-facility networking is thread-free, since insider attacks are always a possibility. What it means is that these sites share the similar level of trustworthiness with regard to the other facilities. For example, certain port scans or pings in the global network may be completely insignificant, whereas they may mean an attack if performed inside the facility. On the other hand, an intruder working from the inside of the facility may have considerably more power than one working from the outside. Note that the first hypothesis is in the direction of considering the facility as a more benign environment, whereas the second is not.

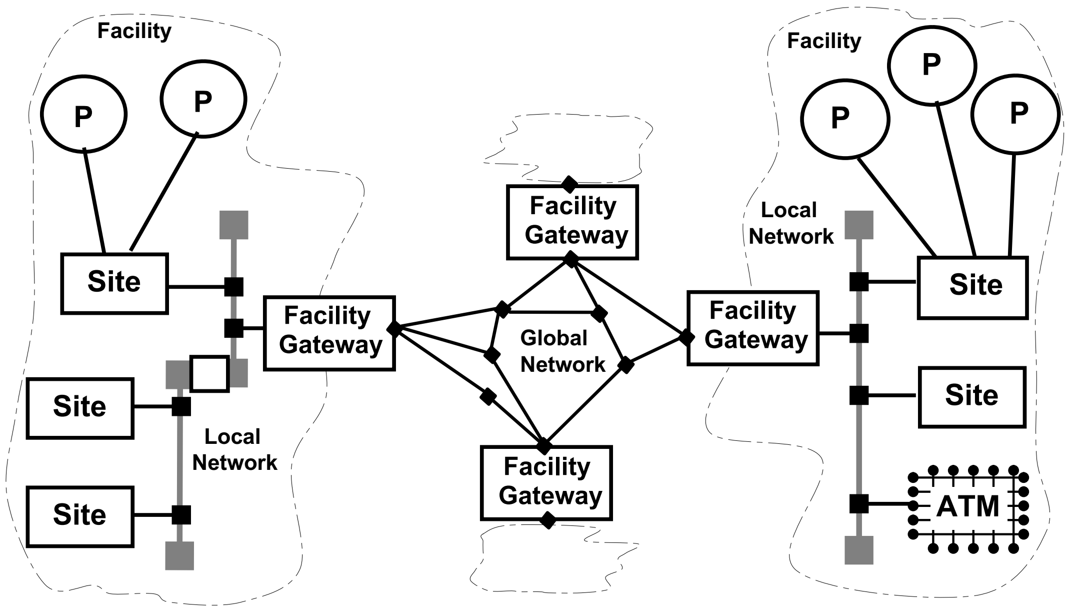

**Figure 3: WAN-of-LAN topology.**

Figure 3 presents a graphical representation of a two-tier WAN-of-LANs. Sites belonging to the same facility are hidden behind a single entry-point, the *facility gateway*, which logically represents the internal nodes for the rest of the network. The facility gateway is a privileged candidate for enforcing the security policy of the organization, making decisions about which traffic can exit or enter in the facility. Therefore, from a security stand point, the facility gateway can: do firewalling tasks, establishing inter-facility virtual private networks; inspecting incoming and outgoing traffic for attack and intrusion detection; internal topology hiding through network address translation, etc. Indeed, the gateways are providing access control mechanisms to implement the protection service (see Section 3.2.2.2). The facility gateways performs also most of the global communication within the organization, tunnelling for instance normal messages and using specific protocols to ensure the right levels of timeliness, reliability, and security.

### 2.3.3 Recursive Application

The WAN-of-LANs topological construct can be applied recursively at different levels, to represent finer grain interactions inside one organization or very-large-scale relations among several institutions (such as the ones involved in the power generation and distribution).

On an intra-facility level, further hierarchies can de defined, namely those already deriving from hierarchical organization of subnetworks and domains, and those resulting from specific responsibilities of certain nodes (e.g., inside a power generation facility there are process controllers nodes and engineering workstations, which rank differently in terms of their criticality).

At an intra-organization level, the topology depicted in Figure 3 can be instantiated to represent an organization with multiple geographically dispersed facilities, interconnected by secure tunnels whose end points are the *internal* gateways. One of the roles of these facility gateways is to implement for instance the Virtual Private Network (VPN) to secure the internal communication. If the organization needs to be connected to the Internet and to other organizations, the same picture can represent the inter-organization interactions. Here, each facility is so to speak the top-level facility, the one containing the organization portal to the Internet. Communication with other organizations through the Internet is ensured through *external* facility gateways.

### 2.3.4 Example Application of the WAN-of-LANs in the Electrical Context

This WAN-of-LAN approach has also been proven successful in previous electricity applications. For instance in the DepAuDE[10] project, the LAN concept corresponds to the intra-site network, while the WAN corresponds to the inter-site network [Deconinck et al. 2003].

An electrical application runs on several nodes of a particular site, interconnected via an *intra-site* (local) network. This intra-site network is dedicated to this application only, and it provides adequate real-time support. The application can interact with applications on other sites over an *inter-site* (wide area) network, via gateway nodes. This inter-site network may be a non-dedicated, open network -such as the Internet- shared with other applications, and hence not under application control.

In DepAuDE, different solutions to more robustness have been applied at both levels.

A gateway node performs *inter*-site communication using tunnelling and *Redundant Source-Routing* (multiple messages over disjoint paths) in order to increase inter-site connection availability and reduce message latency [Mazzini et al. 2002].

At the intra-site level, a script-driven recovery approach was designed and implemented in the middleware, based on a language to express recovery strategies - i.e., to describe diagnosis, containment and recovery actions to be executed when an error is detected. As

[10] DepAuDE project (FP5 IST-2000-25434) Dependability for embedded automation systems in dynamic environments with intra-site and inter-site dependability aspects.

such, it is possible to start a standby task, to reset a node or link, to move a task to another node, to generate synchronization signals for reconfiguration, etc. This language to express recovery strategies has been called ARIEL [De Florio & Deconinck 2002]. ARIEL expresses error recovery in terms of IF guard THEN action [ELSE action] statements. The guard is a Boolean expression resulting in queries to a database of containing the state of the system and its applications. The THEN (ELSE) keyword marks the beginning of a list of fault tolerance actions to be executed when the guard is evaluated as true (false). Three sets of software modules build up the middleware architecture [Deconinck et al. 2002]: a Basic Services Layer (BSL), providing a real-time operating system interface for task handling and inter process communication; a set of Fault Tolerance Mechanisms (FTM), performing error detection, network and task monitoring, fault masking, etc; and a BackBone, i.e., a distributed application that maintains information on application progress as well as on the system topology and status. The BackBone receives event notifications generated by BSL, FTM and the user application, and retains this information in a database. When notified of a detected error, it interprets recovery strategies written in ARIEL by querying the database to assess the system status in order to orchestrate fault tolerance actions.

# 3 ARCHITECTURE

The CRUTIAL architecture encompasses:

- *Architectural configurations* featuring trusted components in key places, which a priori induce prevention of some faults, and of certain attack and vulnerability combinations.
- *Middleware devices* that achieve runtime automatic tolerance of remaining faults and intrusions, supplying trusted services out of non-trustworthy components.
- *Trustworthiness monitoring mechanisms* detecting situations not predicted and/or beyond assumptions made, and adaptation mechanisms to survive those situations.
- *Organisation-level security policies* and access control models capable of securing information flows with different criticality within/in/out of a CII.
- *An application-related service architecture* that is able to perform resource discovery in order to build overlay networks in which electricity services function resiliently.

Some of these aspects build on results from the MAFTIA project[11] in this field [Verissimo et al. 2006], but extend them significantly to attend the specific challenges of the critical information infrastructure problem, for example, timeliness, global access control, and above all non-stop operation and resilience.

Given the severity of threats expected, some key components are built using architectural hybridisation methods in order to achieve *trusted-trustworthy* operation [Verissimo et al. 2006]: an architectural paradigm whereby components prevent the occurrence of some failure modes *by construction*, so that their resistance to faults and hackers can justifiably be trusted. In other words, some special-purpose components are constructed in such a way that we can argue that they are always secure, so that they can provide a small set of services useful to support intrusion tolerance in the rest of the system.

Intrusion tolerance mechanisms are selectively used in the CRUTIAL architecture, to build layers of progressively more trusted components and middleware subsystems, from baseline untrusted components (nodes, networks). This leads to an automation of the process of

[11] Malicious-and Accidental-Fault Tolerance for Internet Applications (FP5 IST-1999-11583). The web site of the project is at *www.maftia.org*.

building trust: for example, at lower layers, basic intrusion tolerance mechanisms are used to construct a trustworthy communication subsystem, which can then be trusted by upper layers to securely communicate amongst participants without bothering about network intrusion threats.

One of the innovative aspects of this work, further to intrusion tolerance, is the resilience aspect, approached through two paradigms: *proactive-resilience* to achieve exhaustion-safety [Sousa et al. 2005], to ensure perpetual, non-stop operation despite the continuous production of faults and intrusions; and *trustworthiness monitoring* to perform surveillance of the coverage stability of the system, that is, of whether it is still performing inside the assumed fault envelope or beyond assumptions made [Bondavalli et al. 2004]. In the latter case, dependable adaptation mechanisms are triggered.

Finally, the desired control of the information flows is partly performed through protection mechanisms using an adaptation of *organisation-based access control* models [Abou El Kalam et al. 2003] for implementing global-level security policies.

The mechanisms and algorithms in place achieve system-level properties of the following classes: trustworthiness or resistance to faults and intrusions (i.e. security and dependability); timeliness, in the sense of meeting timing constraints raised by real world control and supervision; coverage stability, to ensure that variation or degradation of assumptions remains within a bounded envelope; dependable adaptability, to achieve predictability in uncertain conditions; resilience, read as correctness and continuity of service even beyond assumptions made.

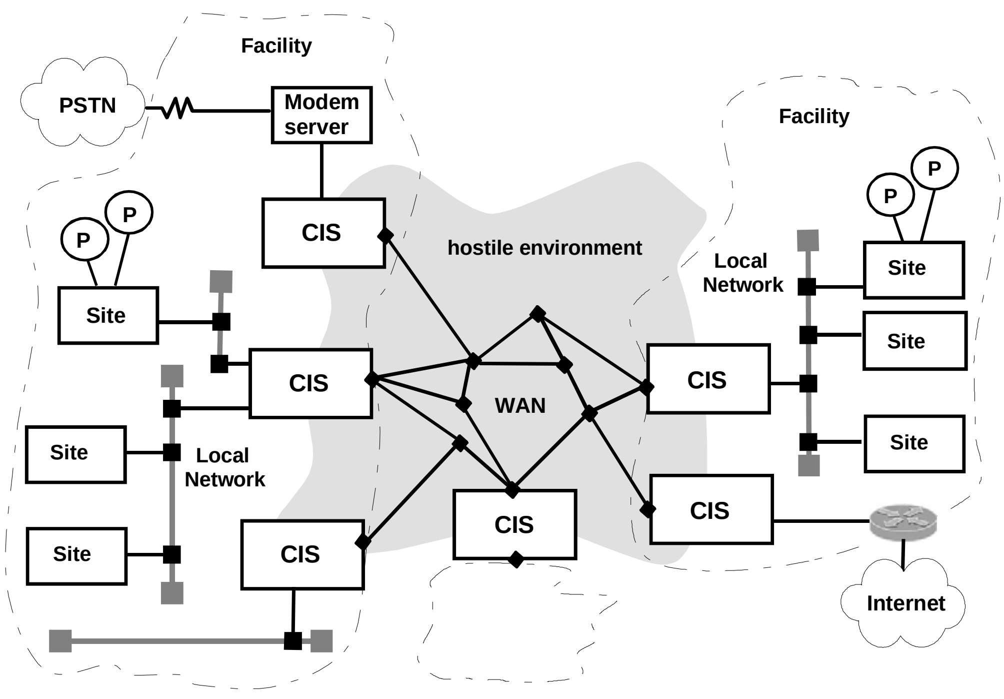

**Figure 4: CRUTIAL overall architecture (WAN-of-LANs connected by CIS, *P* processes live in the several nodes).**

## 3.1 Main Architectural Options

We view the system as a WAN-of-LANs. There is a global interconnection network, the WAN, that switches packets through generic devices that we call *facility gateways*, which are the representative gateways of each LAN (the overall picture is shown in Figure 4). The WAN is a logical entity operated by the CII operator companies, which may or may not use parts of public network as physical support. A LAN is a logical unit that may or may not have physical reality (e.g., LAN segments vs. VLANs[12]). More than one LAN can be connected by the same facility gateway. All traffic originates from and goes to a LAN. As example LANs, the reader can envision: the administrative clients and the servers LANs; the operational (SCADA) clients and servers LANs; the engineering clients and servers LANs; the PSTN modem access LANs; the Internet and extranet access LANs, etc.

The facility gateways of a CRUTIAL critical information infrastructure are more than mere TCP/IP routers. Collectively they act as a set of servers providing distributed services relevant to solving our problem: *achieving control of the command and information flow, and securing a set of necessary system-level properties*. CRUTIAL facility gateways are called *CRUTIAL Information Switches (CIS)*, which in a simplistic way could be seen as sophisticated circuit or application level firewalls combined with equally sophisticated intrusion detectors, connected by distributed protocols.

This set of servers must be intrusion-tolerant (i.e., must tolerate intrusions), prevent resource exhaustion providing perpetual operation, and be resilient against assumption coverage uncertainty, providing survivability. The services implemented on the servers must also secure the desired properties of flow control, in the presence of malicious traffic and commands, and in consequence be themselves intrusion-tolerant.

An assumed number of components of a CIS can be corrupted. Therefore, a CIS is a logical entity that has to be implemented as a set of replicated physical units (CIS replicas) according to fault and intrusion tolerance needs. Likewise, CIS are interconnected with intrusion-tolerant protocols, in order to cooperate to implement the desired services.

## 3.2 System Components

### 3.2.1 Crutial Node

The structure of some of the CII nodes, which we call *CRUTIAL nodes*, can follow the node structuring principles for intrusion-tolerant systems explained in [Verissimo et al. 2003][Verissimo et al. 2006]:

- The notion of *trusted* - versus untrusted - *hardware*. For example, most of the hardware of a CIS is considered to be untrusted, with small parts of it being considered trusted-trustworthy.
- The notion of *trusted support software*, trusted to execute a few critical functions correctly, the rest being subjected to malicious faults.
- The notion of *run-time environment*, offering trusted and untrusted software and operating system services in a homogeneous way.
- The notion of *trusted distributed components*, for example software functions implemented by collections of interacting CIS middleware.

[12] Virtual LANs.

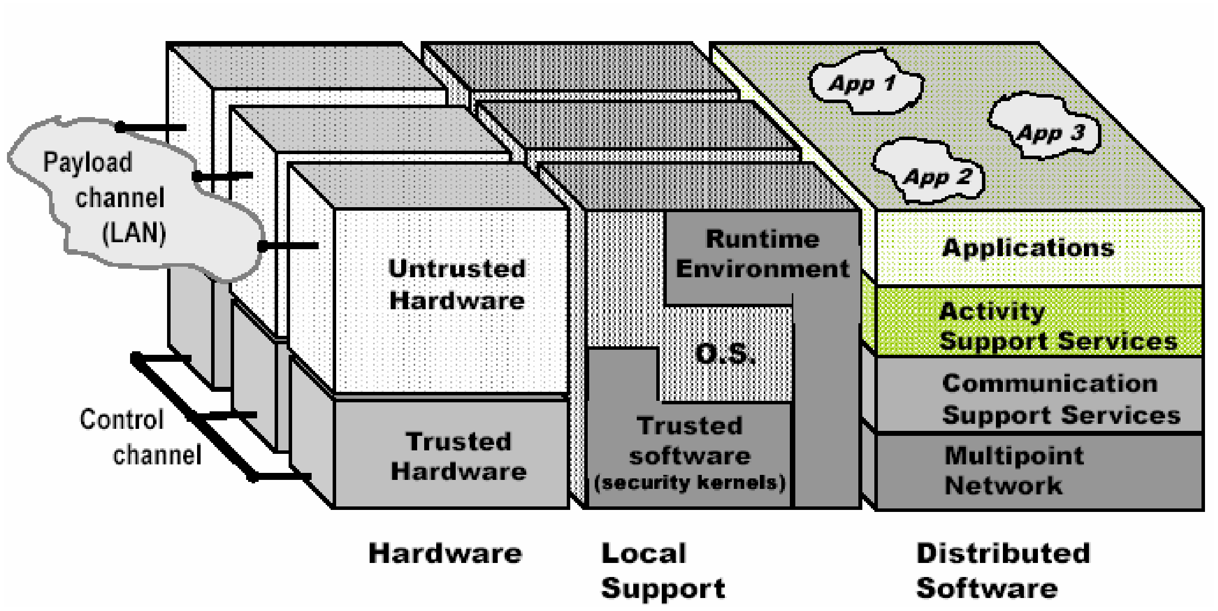

**Figure 5: Architecture and interconnection of CRUTIAL nodes (e.g., CIS).**

Until now, and in the context of this document, we consider only one instantiation of CRUTIAL nodes, the CRUTIAL Information Switch (CIS) nodes. However, other specific nodes, for example, controllers needing to meet high trustworthiness standards, may be also built to a similar structure. A snapshot of the CRUTIAL node is depicted in three dimensions in Figure 5, where we can perceive the above-mentioned node structuring principles.

Firstly, there is the *hardware* dimension, which includes the node and networking devices that make up the physical distributed system. We assume that most of a node's operations run on untrusted hardware, e.g., the usual machinery of a computer, connected through the normal networking infrastructure, which we call the *payload channel*. However, some nodes - CIS, for example - may have pieces of hardware that are trusted, for example, that by construction intruders do not have direct access to the inside of those components. The type of trusted hardware featured in CIS is an *appliance board with processor*, which may or not have an *adapter to a control channel* (an alternative trusted network), as depicted in Figure 5. This appliance is plugged to the CIS's main hardware.

Secondly, services based on the trusted hardware are accessed through the *local support* services. The rationale behind our trusted components is the following: whilst we let a local node be compromised, we make sure that the trusted component operation is not undermined (crash failure assumption).

Thirdly, there is the *distributed software* provided by CRUTIAL: middleware layers on top of which distributed applications run, even in the presence of malicious faults (far right in Figure 5). More about the distributed software used in the CRUTIAL node, i.e., its middleware, will be presented in Section 4.

### 3.2.2 Crutial Information Switches

Let us briefly discuss how CIS are made trusted-trustworthy components. CIS are built with a combination of untrusted and trusted hardware of varying degrees, depending on the needs and criticality of the traffic (sink or source) and the services they support. CIS individual resilience is enhanced by proactive resilience mechanisms, using a construct called

Proactive Resilience Wormhole [Sousa et al. 2005b] (described in Section 4.5) aiming at providing for perpetual execution of a given set of CIS, despite continued intrusion and/or failure of an assumed simultaneous maximum number of CIS at an assumed maximum rate.

These notions can be recursively used to construct a logical CIS which is in fact a set of replicated physical CIS units, running some internal intrusion-tolerant protocols so that the whole appears to the protocol users as a single logical entity sinking/sourcing to/from a given LAN, but is in fact resilient to attacks on the CIS themselves. This is a powerful combination since the resilience of protocols running on such intrusion-tolerant CIS components is commensurate to arbitrary-failure counterparts.

CIS are in addition provided with trustworthiness monitoring subsystems, aiming at assessing the trustworthiness of the CIS itself: as a function of the evolution of the coverage of assumptions underlying the whole fault and intrusion tolerant design. As such, trustworthiness becomes a dynamic property, which provides further resilience to the CIS, through dependable adaptation: automatically reacting to environment uncertainty (changing fault and/or attack levels) and maintaining coverage stability, by changing operation parameters or modes automatically. Finally, for very high levels of resilience, CIS construction and or reconfiguration in the course of proactive recovery may be based on diversity techniques such as n-version programming or obfuscation [Littlewood & Strigini 2004] (diversity is discussed in more detail in Section 4.5).

The desired properties of the (logical) CIS have to be assured using proper methodologies. At a first stage, we plan to test CIS using attack injection techniques [Neves et al. 2006], in which attacks are generated and performed automatically with the purpose of finding vulnerabilities. However, ultimately CIS will have to pass a certification process based on the Common Criteria [ISO 1999].

The CIS provides several services to the SCADA system. The most important services envisioned for CIS are described bellow.

#### 3.2.2.1 LAN Traffic Labelling

A LAN is the top-level unit of the granularity of access control, regardless of possible finer controls. It is also and correspondingly, a unit of trust or mistrust thereof. In fact, we are not concerned with what happens inside a LAN, except that we may attribute it a different level of trust. For instance, if the LAN is a SCADA network, the level of trust is high, but if it is the access to the Internet then the level of trust is low. Traffic (packets) originating from a LAN receives a label that reflects this level of trust, and contains access control information, amongst other useful things. The trustworthiness of a label (that is, the degree in which it can or not be tampered with) can vary, depending on the criticality of the service. In the context of this document, and without loss of generality, we assume it is an authenticated proof of a capacity.

#### 3.2.2.2 Protection Service

One of the main functions of the CIS is to make egress/ingress access control based on LAN packet labels and/or additional mechanisms, implementing an instance of the global security policy. The protection service (PS) is the module of the CIS responsible to provide this functionality.

The PS works mainly like a firewall. It captures packets that pass through the CIS, checks if these packets satisfy the security policy being enforced, and forwards the approved packets, discarding those that do not satisfy the policy. Besides that, several other characteristics of the CIS-PS make it a unique protection device.

- *Distributed firewall*: CIS can be used in a redundant way, enforcing the same policies in different points of the network. An extreme case is to have a CIS in each gateway interconnecting each substation network, and a CIS protecting each critical component of the SCADA/PCS network. This concept of using firewalls to protect hosts instead of only at network borders was introduced in [Bellovin 1999], and protection devices that support it are called Distributed Firewalls. The idea is especially useful for critical information infrastructures given their complexity and criticality, with many routes into the control network that cannot be easily closed (e.g., Internet, dial-up modems, VPNs, wireless access points) [Byres & Lowe 2004].
- *Application-level firewall*: Critical infrastructures have many legacy components that were designed without security in mind, thus do not employ security mechanisms like access control and cryptography [Dzung et al. 2005]. Since these security mechanisms are not part of the SCADA/PCS protocols and systems, which must still be protected, protection must be deployed in some point between the infrastructure and the hosts that access it. The CIS has to inspect and evaluate the messages considering application-level semantics because, as already said, the application (infrastructure) itself does not verify it.
- *Rich access control model*: Besides the capacity to inspect application-level messages, the CIS need to support a rich access control that takes into account the multi-organizational nature of the critical infrastructures as well as their different operational states. Considering Power Systems, there are several companies involved in generation, transmission and distribution of energy, as well as regulatory agencies, and several of these parties can execute operations in the power grid. Moreover, almost all Power System operation is based on a classical state model of the grid [Fink and Carlsen 1978]. In each state of this model, specific actions must be taken (e.g., actions defined in a defense plan to avoid or recover from a power outage), and many of these actions are not allowed in other states (e.g., a generator cannot be separated automatically when the Grid is in its normal state). These two complex facets of access control in critical infrastructures require more elaborated models than basic mandatory, discretionary or role-based access control. To deal with this, in the architecture of CRUTIAL we adopt a more elaborated model, OrBAC (Organization Based Access Control) [Abou El Kalam et al. 2003]. It allows the specification of security policies containing permissions, prohibitions, obligations and recommendations, taking into account the role of the subject that is part of an organization, the action it wants to execute, the target object of this action and the context in which is executed. An example: “In context ‘emergency’, operators from company C can execute maintenance operations on device D.” This model will be discussed more extensively in Section 4.3.
- *Intrusion-tolerant firewall*: As discussed in the introduction, the level of security of current systems connected to the Internet is not adequate for the infrastructures we are concerned with, given their criticality. To improve the security and dependability of the CIS, it is designed to be intrusion-tolerant [Verissimo et al. 2003]: it is replicated in $N$ machines and follows its specification as long as at most $f$ of these machines are attacked and have their behaviour corrupted. To reinforce the resilience of the CIS, we employ proactive recovery to periodically rejuvenate the replicas, guaranteeing perpetual correct operation as long as no more than $f$ replicas become faulty in a giving recovery period [Sousa et al. 2005b] (see Section 4.5).

#### 3.2.2.3 The CIS Communication Services

Another important function of the CIS is to provide resilient and secure communication services for exchanging information between the different CIS-protected realms. The main communication services expected to be supported by CIS are the following:

- *Authenticated reliable point-to-point communication*: This service implements the abstraction of a reliable authenticated point-to-point channel connecting the CIS-protected realms. It is a fundamental service upon which all other dependable communication services will be built. Since many of the potential security problems identified in critical infrastructures are related with Denial of Service attacks, it is important to this service to be resilient to this kind of attack. One possible approach to cope with it is to use application-level overlay routing to deviate traffic from network regions under attack [Andersen et al. 2001][Obelheiro et al. 2006].
- *Byzantine fault-tolerant multicast*: These services can be used to disseminate information and commands between many CIS units, even in presence of malicious nodes on the SCADA network. There are different types of these primitives, such as *total order multicast*, *reliable multicast*, etc. The several flavours of multicast primitives that can be used in the context of CRUTIAL will be discussed in Section 4.2.
- *Timely communication:* Some of the services running in CIS may require some degree of timeliness, given that SCADA implies synchrony, and this is a hard problem in the presence of malicious faults, so we plan to do research towards that direction. This kind of service must provide communication primitives (build upon the previous ones) that can execute message exchange between different CIS with limited and known time bounds with higher probability. Some *Dependable Adaptation* mechanism must be employed to ensure that this probability will stay valid even in face of faults and attacks [Casimiro & Verissimo 2001].

#### 3.2.2.4 Other Services

Besides the services depicted above, other services can be introduced in the CIS as needed. An example of such service is a pattern-sensitive information and command traffic analysis (behaviour and/or knowledge-based intrusion detection) with intrusion-tolerant synchronisation and coordination between local Intrusion Detection Systems (IDSs).

# 4 MIDDLEWARE

The communication among the CIS and possibly other hosts of the critical information infrastructure is supported by a *middleware* being developed in the context of the project. This *CRUTIAL middleware* implements functionalities at different levels of abstraction, as represented in Figure 6. As mentioned earlier, a middleware layer may overcome (through intrusion tolerance) the fault severity of lower layers and provide certain functions in a trustworthy way.

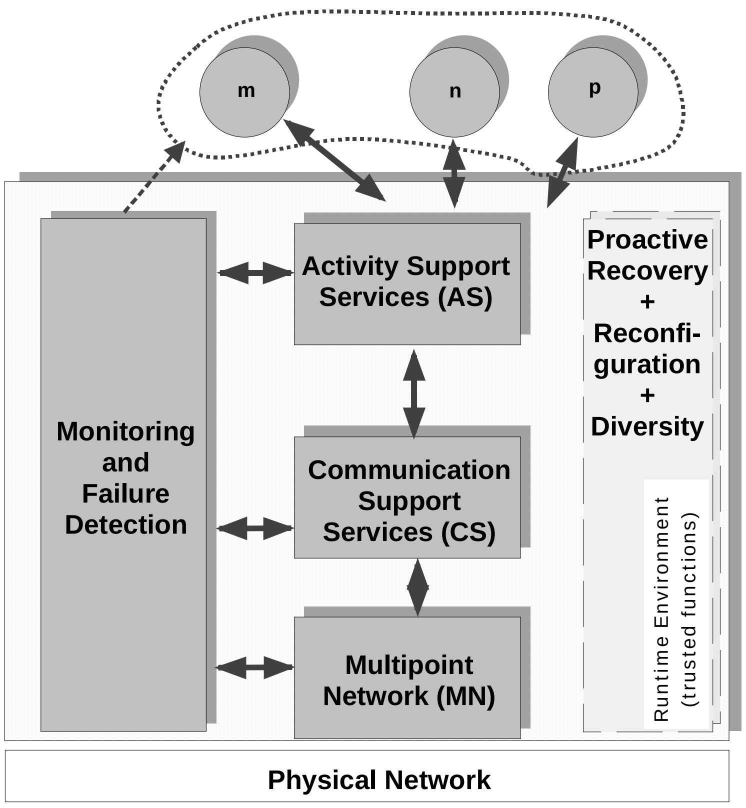
m
n
p
Monitoring and Failure Detection
Activity Support Services (AS)
Communication Support Services (CS)
Multipoint Network (MN)
Proactive Recovery + Reconfi-guration + Diversity
Runtime Environment (trusted functions)
Physical Network

**Figure 6: CIS middleware.**

The lowest layer in Figure 6 is the *Multipoint Network* module (MN), created over the physical infrastructure. Its main properties are the provision of multipoint addressing, secure channels (IPsec, SSL, etc.), and management functions, hiding the specificities of the underlying network.

The *Communication Support Services* module (CS) implements basic cryptographic primitives, Byzantine agreement, consensus, group communication and other core services. The CS module depends on the MN module to access the network.

The *Activity Support Services* module (AS) implements building blocks that assist a participant activity, such as replication management (e.g., state machine, voting), IDS and firewall support functions, access control. It depends on the services provided by the CS module.

The block on the left of the figure generically implements Monitoring and Failure Detection. Failure detection assesses the connectivity and correctness of remote nodes, and the liveness of local processes. Trustworthiness monitoring and dependable adaptation mechanisms also reside in this module, and have interactions with all the modules on the right. Both the AS and CS modules depend on this information.

The block to the right represents the support services. These include the usual operating system's services, but also the trusted services supplied in support to the algorithms in the various modules: proactive recovery, reconfiguration, and diversity management.

All the blocks in Figure 6 are described in the following subsections.

## 4.1 Multipoint Network

The lowest middleware layer – *Multipoint Network (MN)* – provides basic communication services. They are "basic" in the sense that they support upper layer protocols and are provided by standard protocols, like IP, IPsec and TCP. This section presents briefly some of the protocols that can be provided in the MN module, albeit several others are available. The presentation is organized in terms of the layer of the TCP/IP reference model to which these protocols belong: Physical, Connection, Network, Transport, and Application. We skip the Physical and Connection layers for which there are many technologies and are too low level to be considered middleware: Ethernet (wired and wireless), SDH/Sonet/ATM, Frame Relay, copper circuits, etc. Application layer protocols, like some protocols specific for critical infrastructures and industrial systems (MMS and ICCP) are also not described since they are at a higher level than the middleware. Therefore, the layers we are interested in this section are the Network layer and the Transport layer.

### 4.1.1 Network Layer

The main service provided by the Network layer in the Internet is routing data packets, called datagrams, from the source host to the destination host. Hosts are interconnected by nodes called routers that inspect the datagrams to forward – or route – them to the next router or the destination.

The format of the datagrams is defined by the most widely used Network layer protocol in the Internet, appropriately called the *Internet Protocol (IP)* [Postel 1981a]. Nowadays, IP underlies most communication networks around the world, including the Internet, corporate networks and even some control networks, so it is important to give some insight about it.

The most important pieces of data in an IP datagram are the source and destination host addresses. Hosts or, more precisely, each host's interface is identified by an IP address, which has 32 bits in IPv4, the current almost universal version of IP adopted. A shift for IPv6 is expected, but truth be said it is being expected for many years now. IP also provides other services like fragmentation and reassembly of packets too big for the size of the packet transported by the physical network. IP does not ensure the reliability of the communication.

IP gives the ability to transmit a datagram to a group of hosts, identified by a single IP destination address, of a range of addresses reserved for that purpose. This is important for CRUTIAL middleware since it involves multicast to several hosts, e.g., for several CIS. This form of using IP is usually called *IP multicast*. The multicast datagram is delivered to all members of the group with the same guarantees given by the regular IP datagrams: it is not guaranteed to reach all members, it is not guaranteed to arrive intact to all members and it is not guaranteed to arrive in the same order to all members, relative to other datagrams.

Hosts can join and leave the group at any time, i.e., group membership is dynamic. Not only can a host belong to more than one group at a time, but it also does not need to be a member of a group to send datagrams to it. A group may be permanent or transient. A permanent group has a well-known, administratively assigned IP address. It is the address, not the membership of the group, that is permanent; at any time a permanent group may have any number of members, even zero. Those IP multicast addresses that are not reserved for permanent groups are available to be dynamically assigned to temporary groups, which exist only as long as the group has members in it.

Multicast groups cannot span the whole Internet since not all routers support this functionality. In fact, only certain, typically small, "islands" of routers support it (e.g., certain

Internet Service Providers support it, while others do not). There is, however, a large island that supports multicast and spans several continents, called the MBone.

## IPSec

We already mentioned that IP does not ensure the reliability of the communication, i.e., it can loose messages. However, IP also does not ensure the communication security. This means that messages can be modified and its content read by anyone with access to the network, e.g., a hacker controlling a router. To deal with this problem, there is a security extension to IP called IPsec [Kent & Atkinson 1998a][Kent & Atkinson 1998b][Kent & Atkinson 1998c]. IPSec has an important role in CRUTIAL since it is a basic mechanism to ensure security in the Network layer, so this section gives an introduction to the protocol.

IPSec is designed to enhance the security of IPv4, providing interoperable, high-quality, cryptographically-based security. It offers several services, such as access control, connectionless integrity, data origin authentication, protection against replays, confidentiality (through encryption) and limited traffic flow confidentiality. Since these services are offered at the Network/IP layer, they can be used by any higher layer protocol, such as TCP, UDP, HTTP, etc. IPSec also supports negotiation of IP compression [Shacham et al. 1998], motivated by the observation that encryption used within IPSec prevents effective compression by lower protocol layers.

IPsec is divided in two basic (sub)protocols:

- Authentication Header (AH) – provides connectionless integrity, data origin authentication, and an optional anti-replay service.
- Encapsulation Security Payload (ESP) – provides payload confidentiality (using encryption) and limited traffic flow confidentiality. Optionally, it may also provide connectionless integrity, data origin authentication, and an anti-replay service. Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols.

These protocols may be applied alone or in combination with each other to provide the desired set of security properties in IP. They support two different modes of operation:

- Transport mode – in this mode, IPsec essentially protects upper layer protocols (e.g., TCP).
- Tunnel mode – in this mode, the protocols are applied to tunnelled IP packets, i.e., the IP datagrams themselves are sent through a secure tunnel.

IPSec allows the user or the system administrator to control the granularity at which a security service is offered, allowing, for example, the creation of a single encrypted tunnel to carry all the traffic between two security gateways or a separate encrypted tunnel for each TCP connection between a pair of hosts communicating across these gateways. IPsec can be configured to protect only the integrity of the communication (preventing modifications) or the integrity and the confidentiality of the traffic.

Currently most IPSec implementations do not have an API that can be used by applications to transmit secure data. Instead, it works at the operating system level, and can only be configured by the system administrator. A system administrator can define the policy for IPSec on a host basis, determining the ways by which a host can connect securely to another.

### 4.1.2 Transport Layer

IP solves the problem of end-to-end communication between two hosts. However, the problem we really want solved is slightly different: end-to-end communication between applications. This is the problem solved by the Transport layer. In IP-based networks hosts are identified by an IP address; inside a host, an application is identified by a *port* or, more precisely, by one or more ports, which are 16-bit numbers (range 0-65535). The standard Transport layer Internet protocols are the *User Datagram Protocol (UDP)* and the *Transmission Control Protocol (TCP)*. Both protocols are used to support communication in critical infrastructures, so both are relevant for the CRUTIAL middleware.

UDP provides a datagram mode of packet-switched computer communication in an interconnected set of computer networks [Postel 1980]. Applications can send messages to other programs with a minimum set of guarantees using UDP. The key characteristics of the protocol are: it is transaction oriented, the delivery of messages is not ensured, nor is the order of message arrival, and there might be duplication of messages. UDP in fact is a thin protocol on top of IP, which does not provide more guarantees, only adds information about the source and destination applications, i.e., the source and destination ports.

TCP, on the other hand, is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols supporting multi-network applications [Postel 1981b]. Applications can send data using TCP, in a reliable way, to other programs on host computers attached to distinct but interconnected computer communication networks. TCP does not rely on the protocols below for reliability, but rather assumes that it can obtain a (potentially) unreliable datagram service from the lower level protocols, typically IP. A TCP connection serves to send a stream of data (not independent datagrams), which in practice is split in TCP segments. Reliability means that segments are delivered in the order they were sent and unmodified. In practice, these properties are ensured using a CRC code to detect modifications, and retransmissions to recover from missing or corrupted segments. A disruption of the network can interrupt the delivery of the stream of data if a timeout causes TCP to break the connection. TCP segments include the source and destination ports.

The CRC code in TCP segments is useful to detect accidental modifications, e.g., due to line noise. However, in terms of security it does not protect the segments since a malicious hacker can modify the segment plus the CRC code to fit the segment modification. Malicious modifications have to be detected using Message Authentication Codes (MAC), like those provided by IPsec. In fact, reliable and secure end-to-end application communication can be implemented using TCP over IPsec.

TCP is a complex protocol, with several other mechanisms not discussed here. Examples are flow control (to prevent segments from being sent when the reception buffer has no space), slow start (to avoid contributing to network congestion when a TCP connection is established) and fast retransmit (to cause an earlier retransmission of missing segments).

The *de facto* standard programming interface for TCP and UDP is the Berkeley Unix Socket API [Stevens 1990]. It provides primitives like *socket* (to create a communication endpoint), *bind* (to associate an IP address and a port to a socket), *send* and *sendto* (to send data), *receive* and *receivefrom* (to receive data) and *select* (to block waiting for receiving from several endpoints).

SSL

The secure socket layer (SSL) [Frier et al. 1996], later standardized as Transport Layer Security (TLS) [Dierks & Allen 1999], adds security to TCP. It basically provides peer authentication, and confidentiality and integrity of the communication.

SSL/TLS is a modification of TCP. The initial handshaking is followed by a negotiation of the cryptographic algorithms to use and the establishment of a session key. Authentication is based on public-key cryptography and digital certificates, and can be mutual (both peers authenticate themselves), one-way or simply not done. Integrity and (optionally) confidentiality of data are guaranteed using the session key, respectively by adding a MAC and encrypting the data.

The security guarantees provided by SSL/TLS are similar to those provided by TCP over IPsec, except for the more powerful authentication scheme and the usual availability of a user-level API, something that is not common with IPsec.

## 4.2 Communication Support

The *Communication Support Services* module (CS) provides security and Byzantine fault tolerance primitives, like Byzantine agreement, reliable multicast, and view-synchronous atomic multicast, which enable, for instance, the construction of intrusion-tolerant services like a replicated CIS. The CS module depends on the MN module for basic communication. For instance, Byzantine agreement can be implemented over secure channels provided by TCP over IPsec, or by SSL/TLS. All these protocols aim to be used in an environment prone to malicious attacks and intrusions, so they are Byzantine fault-tolerant or intrusion-tolerant. In other words, they behave as expected even if some of the processes that execute them behave maliciously trying to break the protocol properties.

The CRUTIAL middleware and, in this case, specifically the CS module, provides primitives for multiparty communication and computation. Therefore, the primitives support *group communication*. Groups of processes or hosts can be open or closed. An open group model permits arbitrary hosts to send messages to the group, while in a closed model only hosts which are already members of the group may so communicate. In this section we consider only closed groups, since sending messages to the groups is usually a simpler problem that involves sending them to a subset of the group members.

Groups can also be static or dynamic. In a static group the membership does not change over time, or changes at a very long time scale, such as only upon manual reconfiguration. On the contrary, dynamic groups allow hosts to join a group, leave it, or be excluded if they are faulty (e.g., if they crashed or behaved maliciously). The CRUTIAL middleware leaves it to the applications programmers to decide how to implement either open or closed groups by combining the primitives below. For instance, to implement a dynamic group, a membership protocol (Section 4.2.3) and view-synchronous communication protocols (Section 4.2.4) are needed.

The primitives provided by the CS module can be divided in several classes. Below we present them organized in consensus primitives, static group communication primitives, membership, and dynamic group communication primitives.

### 4.2.1 Consensus Primitives

Byzantine consensus – or Byzantine agreement – is the problem of reaching agreement on one of the values proposed by each process. It is a classical distributed systems problem, first introduced in [Pease et al. 1980], with a great practical interest, since several other distributed systems problems are reducible or equivalent to it [Hadzilacos & Toueg 1994][Cachin et al. 2001][Correia et al. 2006a]. For instance, the problem of atomic broadcast ends up being equivalent to the problem of reaching consensus about the message that have to be delivered and the order in which they have to be delivered.

The relations between consensus and other distributed problems are important because many results stated for consensus automatically apply to these other problems. The most important of those results is probably the impossibility of solving consensus deterministically in an asynchronous system if a single process can fail, even if accidentally by crashing, often called the FLP result [Fischer et al. 1985]. This result lead to a large number of works that slightly modify the system model in such a way that consensus becomes solvable. These techniques to "circumvent" FLP can be classified in four types: (1) those that sacrifice determinism, leading to probabilistic (or randomized) algorithms; (2) those that add time to the model; (3) those that enrich the system model with an oracle (failure detector, wormhole); and (4) those that enrich the problem definition. It is important to understand that consensus algorithms in practice have to use one of these techniques, since each one has its drawbacks.

The problem of consensus can be more formally defined this way. Let us say that a process is *correct* if it follows its algorithm until completion, otherwise it is said to be *faulty*. Each process proposes a value v (or has an initial value v) and decides on the same or a different value. A consensus protocol is one that satisfies the following properties:

- *Validity:* If all correct processes propose the same value v, then any correct process that decides, decides v.
- *Agreement:* No two correct processes decide differently.
- *Termination:* Every correct process eventually decides.

There are several variations of this basic definition, though. If the values are Boolean, i.e., belong to the set {0,1}, the problem is said to be *binary consensus*. If the values belong to a set of values with arbitrary length then the problem is said to be *multi-valued consensus*. This latter version of the problem can be modified in such a way that the value decided in no longer a value but a vector of values, leading to *vector consensus.* More formally, this problem is defined as before except for a new Validity property:

- *Vector validity:* Every correct process that decides, decides on a vector V of size n:
  - o for any process $p_i$, if $p_i$ is correct, then either V[i] is the value proposed by $p_i$ or $\perp$ (which is some value outside the range of valid values); and
  - o at least (f +1) elements of V were proposed by correct processes.

A somewhat different kind of definition is the one used in the *Paxos algorithms* [Lamport 1998][Lamport 2001]. The idea is that processes play one (or more) of the following roles: *proposers* (propose values), *acceptors* (choose the value to be decided) and *learners* (learn the chosen value). The problem can be defined in terms of five properties:

- Only a value that has been proposed may be chosen.
- Only a single value may be chosen.
- Only a chosen value may be learned by a correct learner.
- Some proposed value is eventually chosen.
- Once a value is chosen, correct learners eventually learn it.

Several Byzantine fault-tolerant binary consensus protocols have been given in the literature. Protocols based on randomization, i.e., that at some point pick a random number, and terminate probabilistically were designed, e.g., [Ben-Or 1983][Rabin 1983][Bracha 1984][Cachin et al. 2000]. *Transformations* from binary consensus to multi-valued consensus, vector consensus, atomic multicast and other protocols were presented in [Cachin et al. 2001][Correia et al. 2006a]. Randomized protocols were recently evaluated and compared in [Moniz et al. 2006a][Moniz et al. 2006b].

Several multi-valued consensus protocols are available in the literature, for instance, based on partial-synchrony (adding time to the system model) [Dwork et al. 1988], using failure detectors [Malkhi & Reiter 1997][Kihlstrom et al. 2003][Doudou & Schiper 1997][Baldoni et al. 2003][Doudou et al. 2002][Friedman et al. 2005b], and wormholes [Correia et al. 2005]. Vector consensus protocols are not so common, but a few are available also [Neves et al. 2005][Correia et al. 2006a].

Finally, several Byzantine Paxos algorithms are also available [Castro & Liskov 2002][Martin & Alvisi 2005][Zielinski 2004][Ramasamy & Cachin 2005].

### 4.2.2 Static Group Communication

Consensus primitives are extremely useful and extremely used in distributed systems, but they are more interesting for coordination or as building blocks than for communication. Reliable group communication is based on primitives like reliable multicast and atomic (or total order) multicast.

*Reliable multicast* can be defined in terms of two properties: (1) all correct processes deliver the same messages; (2) if the sender is correct then the message is delivered. There are just a few implementations available [Bracha 1984][Correia et al. 2002]. The *echo multicast* primitive is a weaker and more efficient version of the reliable multicast. Its properties are somewhat similar, however, it does not guarantee that all correct processes deliver a multicasted message if the sender is corrupt [Toueg 1984][Reiter 1994]. In this case, the protocol only ensures that the subset of correct processes that deliver will do it for the same message.

An *atomic multicast protocol* delivers messages in the same order to all processes. It can be seen as a reliable multicast protocol that also orders the messages delivered. However, in fact, atomic multicast is a harder problem, equivalent to consensus. Several atomic multicast protocols are available in the literature, based on: failure detectors [Kihlstrom et al. 2001][Doudou et al. 2002][Reiter 1994], wormholes [Correia et al. 2006b][Correia et al. 2004], randomization [Moser & Melliar-Smith 1999][Cachin et al. 2001][Correia et al. 2006a]. Atomic multicast can be easily implemented on top of consensus.

Other group communication primitives are also available. *FIFO multicast* is a primitive that imposes FIFO order: if a process multicasts a message M before M' then all correct processes deliver M before M'. *Causal multicast* delivers messages according to the relation of potential causality. A message M precedes, or is potentially causally related to, a message M' ($M \rightarrow M'$) iff: (1) a process sends M' after M; or (2) M is delivered to the sender of M' before it sends M'; or (3) there is a message M" such that $M \rightarrow M''$ and $M'' \rightarrow M'$. Causal multicast however to the best of our knowledge has not been used in the context of intrusion tolerance, or Byzantine fault tolerance, probably because it is not clear how to implement it.

### 4.2.3 Membership

On the contrary to static groups, dynamic groups allow members to join and leave the group during normal operation, and those group membership changes should be transparent to the applications using the group's services. Thus, the communication services above are not affected by membership changes, and the focus of this section is on the orthogonal question of how new members join the group and current members leave the group.

The membership of a group at a given instant is called a *view*. The concept was defined in the context of the *virtual synchrony group semantics,* later called *view synchrony* [Birman & Joseph 1987a][Birman & Joseph 1987b]. Informally, view synchrony provides membership information to processes in the form of views and guarantees that all processes that install two consecutive views deliver the same set of messages between these views. However, the architecture of CRUTIAL middleware does not impose a specific semantics and views can be seen simply as the membership at a given moment (although it is possible to use view synchronous primitives; see next section). When views are used, when there is a view change it is multicasted atomically to all group members when they change, in a special system management atomic broadcast stream. Changes can be due to member joins, leaves and to failures. Failures have to be detected using a failure detector.

Wide-area networks are prone to link failures and other communication fluctuations. These effects can lead to network partitions, i.e., to the virtual separation of the network in several sub-networks that are temporarily unable to communicate. This may cause the temporary division of a group in two or more subgroups. To handle this type of failures, the CRUTIAL middleware uses a *primary partition model*, in which at most one of the subgroups is allowed to make progress [Birman 1997]. Processes belonging to the rest of the subgroups are eventually removed from the primary partition subgroup. We have to use the primary partition model because most group communication protocols can only terminate with the contribution of a majority of the processes (typically more than two thirds), so if the group is split in several subgroups these protocols can only terminate in at most one of the subgroups.

There are a few membership services in the literature that might be used as part of the CRUTIAL middleware, except that all were designed for LANs only, and probably do not work well in WANs: Rampart [Reiter 1996][Ramasamy et al. 2002], SecureRing [Kihlstrom et al. 2001], SecureGroup [Moser et al. 2000] and Worm-IT [Correia et al. 2006b].

### 4.2.4 Dynamic group communication

Communication in dynamic groups can be made using the same protocols as those that are used for static groups, presented in Section 4.2.2. However, applications based on dynamic groups typically have additional requirements, like messages should be delivered only to group members, and messages have to be delivered to all group members. More precisely, often the communication is required to satisfy the following property*:*

- *View synchrony:* It two correct processes in group G install views $V_n$ and $V_{n+1}$, then both processes deliver the same messages in view $V_n$.

The implementation of this property usually requires a *stabilization protocol*. When a new view, say $V_{n+1}$, is to be installed, the stabilization protocol makes agreement in what messages are yet to be delivered by the group communication primitives (e.g., reliable multicast or atomic multicast) in view $V_n$ and forces these messages to be delivered before $V_{n+1}$ is installed.

View-synchronous reliable and atomic multicast Byzantine fault-tolerant primitives for LANs were presented in [Reiter 1994][Kihlstrom et al. 2001][Moser & Melliar-Smith 1999][Correia et al. 2006b][Moser et al. 2000].

## 4.3 Activity Support

In this section, we will describe the access control activity support service. In the future, and as the project progresses, other services might be added when needed, such as replication management (e.g., state machine replication, voting) and firewalling support functions.

We will start by first reminding the main characteristics of a CII (*Critical Information Infrastructure*), with respect to security requirements and needs, focussing in particular on the electricity and associated information and communication infrastructures considered in the context of CRUTIAL. Then, we discuss the limits of traditional access control models and we extend them to cover the particularities of CIIs. Finally, we suggest an access control model that supports these characteristics: the PolyOrBAC (Poly-*Organization based Access Control Model*).

In particular, PolyOrBAC handles the collaboration between the CII's organizations, while controlling whether the interactions between these organizations comply with their expected behaviour. In fact, each component or subsystem of the CII could have its own security objectives and should be able to cooperate with the other components and subsystems.

For this aim, PolyOrbac extends OrBAC (*Organization based Access Control Model*) [Abou El Kalam et al. 2003] and uses web-services-based mechanisms. In particular, we will describe how the CII's homogeneous security policy (based on OrBAC) can be implemented on the CIS "*CRUTIAL Information Switches*" (through the web services interfaces) by using XML-based security mechanisms.

### 4.3.1 Problem statement

As described previously, the CRUTIAL architecture can be seen as a group of interconnected systems and LANs):

- A CII is a WAN connecting several organizations involving different actors and stakeholders (e.g., power generation companies, energy authorities, external maintenance service providers, transmission and distribution operators centres, etc.) and various LANs (see Figure 7 -- more detailed information is provided in deliverable D2 [Crutial_D2 2007]).
- LANs are composed of one or more logical and physical systems and are interconnected through specific CIS nodes. Each LAN (and CIS) proposes its services to other systems and has its own applications and access control policy.
- CIS are hardware and/or software components (e.g., routers, gateways, firewalls, IDS, servers) that represent gateways of each LAN.

The CRUTIAL architecture can thus be seen as several organizations interconnected through CIS. These organizations can be power generation companies, power plants, Transmission System Operator (TSO), Distribution System Operator (DSO), energy authorities, etc. Each organization contains its own resources, services, applications, operating system, functioning rules, security objectives and security rules.

Moreover, to cope with the changing parameters and surrounding environment, the CII structure must be flexible and extensible. In fact, a CII can be extended over several geographic areas (or countries). Furthermore, as we have an opened and collaborative system, some of these organizations are accessible by both internal and external users.

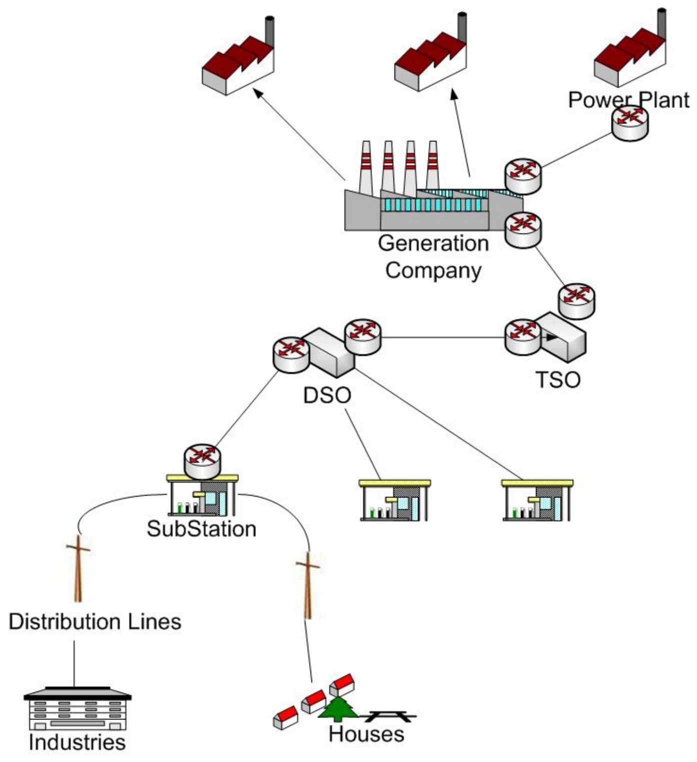

**Figure 7: Simplified architecture of an electric power grid.**

We believe that, in order to provide a controlled cooperation (adapted to CII), each CIS should contain efficient security mechanisms enforcing the local security policy as well as collaboration security policies. These policies and mechanisms should be able to prevent accidental non-authorized accesses as well as malicious accesses.

Authorization aims at allowing legitimate action, while forbidding non-registered users to carry out any action in the system as well as forbidding registered users to carry out non-authorized actions.

In order to define authorized and prohibited actions, we should establish a security policy. The security policy is defined in the Common Criteria as *the set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information* [ISO 2005].

Basically, a security policy[13] is specified through:

- The security objectives that must be satisfied, e.g., “submissions to call for tenders must remain secret (i.e., must not be disclosed to competing organizations)”;

[13] Actually, a security policy could be developed according to three different axes: *physical* (e.g., measures taken against theft and disasters), *administrative* (e.g., separation of duties) and *logical*. The latter concerns the access control functions (e.g., authentication and authorization). As in this WP we only deal with authorization policies, we use “security policy” or “access control policy” to designate “authorization policy”.

- The rules expressing how the system may evolve in a secure way, e.g., "the organization owner of a resource (e.g., a file) is allowed to grant access rights on the resource to other cooperating organizations".

Nevertheless, the security policy, by itself, does not guarantee a secure and correct functioning of the system. Generally, in order to verify its consistency, the security policy is associated to a formal model, which helps to abstract the security policy, to manage its complexity, to detect and resolve conflicting situations, and to check that all the security objectives are covered by the measures already identified.

In addition, the authorization policy should be implemented by suitable security mechanisms (e.g., access control lists "ACL", capacities, filtering rules on firewalls, cryptographic transformation, etc.). In practice, these mechanisms can be selected (e.g., using a commercial-off-the-shelf product) and configured, or specified, if they require a specific development.

In the next section, we present a brief description of "traditional" access control models and we will show that they are not suitable to CII. Afterwards, in Section 4.3.2.1, we present the OrBAC model, in Section 4.3.2.2 the web services, and in 4.3.2.3 how OrBAC can be extended to PolyOrBAC to be applied to CII web services.

### 4.3.2 Security policies and models

Two main classes of access control policies and models have been proposed: discretionary access control (DAC) [NCSC 1987] and mandatory access control (MAC) [Lamspon 1971][Bell & LaPadula 1976][Harrison et al. 1976]. Unfortunately, they do not cover all the identified needs of CIIs. For instance, the HRU model represents the relationships between the subjects, the objects and the actions by a matrix *M* [Harrison et al. 1976]. *M*(*s*, *o*) represents the actions *that* a subject *s* is allowed to carry out on *o*. It is thus necessary to enumerate all the triples (*s*, *o*, *a*) that correspond to permissions defined in the security policy. Moreover, when new subjects, objects or actions are added in or removed from the system, it is necessary to update the policy. HRU is thus not adapted to CRUTIAL, as the number of rules to be specified in the security policies could be very large, due to the fact that a large number of actors, stakeholders and different circumstances need to be distinguished.

Role Based-Access Control (RBAC) [Ferraiolo & Kuhn 1992] is more flexible. Roles are assigned to users, permissions are assigned to roles and users acquire permissions by playing roles [Guiri 1995]. Hierarchical RBAC [Sandhu et al. 1996][Shandhu 1998] adds a requirement for supporting the role hierarchies, while constrained RBAC [Sandhu et al. 1996][Shandhu 1998] enforces the separation of duties. If users are added to the system, only the instances of the relationship between the users and the roles are updated. Moreover, RBAC is deliberately policy neutral. In applying RBAC to a system, the interpretation of permissions is an important step to perform. Unfortunately, RBAC is not directly able to manage the collaboration between independent organizations within a CII.

The OrBAC (Organization-based Access Control) model is an extension of RBAC that details permissions, prohibitions and obligations while remaining implementation independent [Abou El Kalam et al. 2003][Cuppens 2004][Miege 2005]. The main idea is to express the security policy with abstract entities only, and thus to completely separate the representation of the security policy from its implementation. Indeed, OrBAC structure the subjects, the objects and the actions into roles, views, activities. These abstract entities are extensions of the roles defined in RBAC [Ferraiolo & Kuhn 1992], the views used to control access to databases and

XML documents [Bertino et al. 2000][Gabillon 2004] and the tasks introduced in TBAC [Thomas & Sandhu 1994][Thomas & Sandhu 1997].

Let us remind that our aim is to develop protection mechanisms, including for authentication and authorization, adapted to the CRUTIAL architecture, the various organizations involved in the infrastructure and the various roles of the users belonging to these organizations. These mechanisms should be able to enforce a global security policy, defined from the security policies of the various organizations.

Globally, there are two ways to fulfil this objective:

- The first one consists in defining a global and centralized security policy. Actually, this proposal is not interesting in CRUTIAL since a CII involves the cooperation between different organizations, possibly mutually suspicious, with different features, functioning rules and security policies.
- The second one consists in handling the collaboration between the CII subsystems while keeping some local self-determination (for each subsystem). We believe that this proposal is more interesting for CIIs. OrBAC as well as security mechanisms based on web services seem complementarily suitable to give shape to this idea.

In the next subsections, we first summarize OrBAC and we extend it to overcome its limits and to cover the particularities of CIIs. In particular, we will couple an OrBAC extension with web services based security mechanisms. For more details about access control models, please refer to the annex.

#### 4.3.2.1 The OrBAC model

In OrBAC, an organization is a structured group of active entities, in which subjects play specific roles. An activity is a group of one or more actions, a view is a group of one or more objects, and a context is a specific situation that conditions the validity of a rule (see Figure 8).

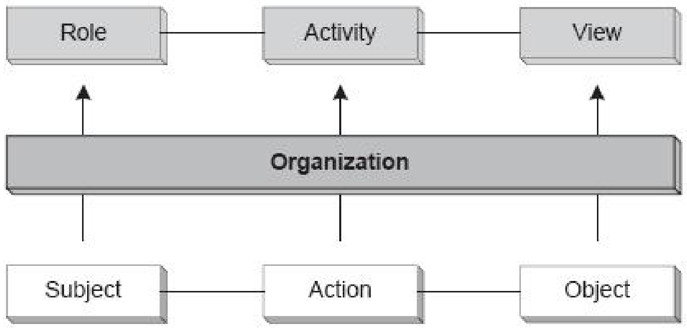

**Figure 8: Abstraction of the traditional Access Control entities.**

Actually, the Role entity is used to structure the link between the subjects and the organizations (see Figure 9). The relationship Empower (org, r, s) means that org employs subject s in role r. In the same way, the objects that satisfy a common property are specified through views (see Figure 10), and activities are used to abstract actions (see Figure 11).

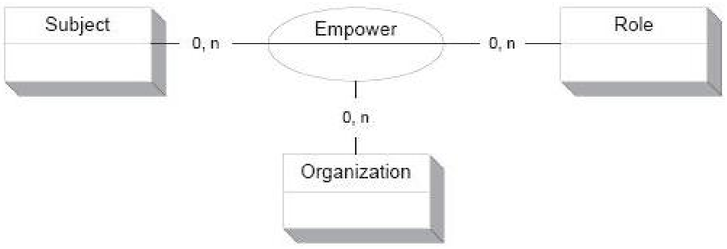

**Figure 9: Abstracting subjects.**

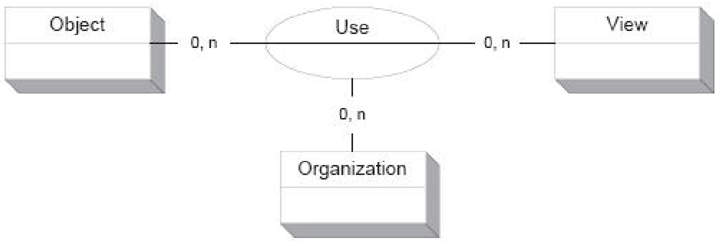

**Figure 10: Abstracting objects.**

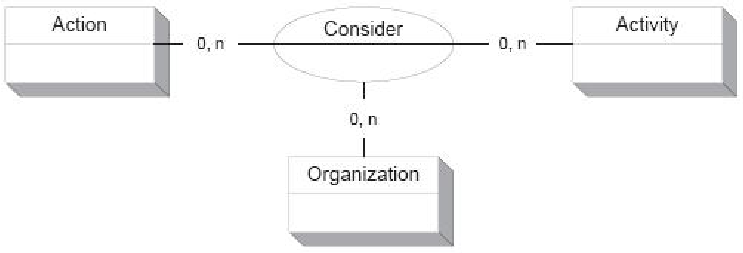

**Figure 11: Abstracting actions.**

Security rules have the following form: *Permission (org; r; v; a; c)*, *Obligation (org; r; v; a; c)*, and *Prohibition (org; r; v; a; c)*. In the context "*c*", organization "*org*" grants role "*r*" the permission (or the obligation or the prohibition) to perform activity "*a*" on view "*v*".

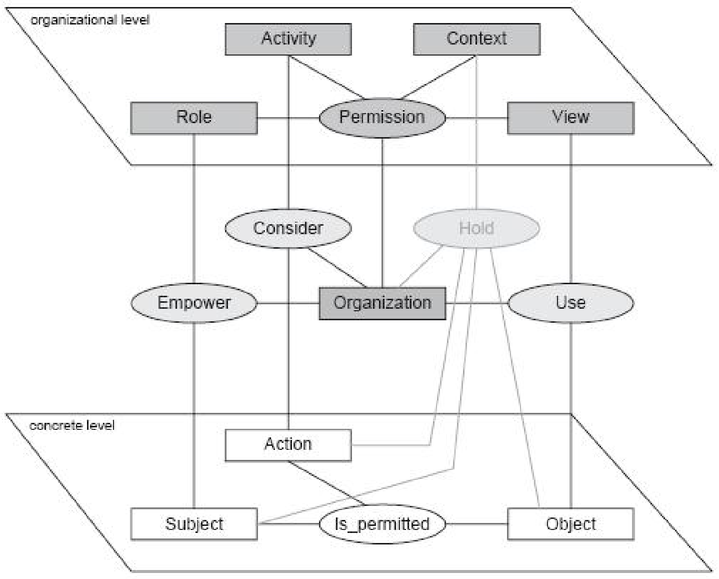

**Figure 12: The OrBAC model.**

As rules are expressed only through abstract entities, OrBAC is able to specify the security policies of several collaborating and heterogeneous organizations (as it is the case for CIIs).

In fact, the same role "operator" can be played by several users belonging to different organizations; the same view "TechnicalFile" can designate TF-Table or TF1.xml (according to the organization); and the same activity "read" could correspond in a particular organization to a "SELECT" action (if the organization has a database system) while in another organization it may specify a OpenXMLfile() action.

Two security levels can be distinguished in OrBAC, abstract and concrete levels (see Figure 12):

- *Abstract level*: the security administrator defines security rules through abstract entities (roles, activities, views) without worrying about how each the organization implements these entities.
- *Concrete level*: when a user requests an access, concrete authorizations are granted (or not) to *him* according to the concerned rules, the organization, the played role, the instantiated view / activity, and the current parameters.

The derivation of permissions (i.e., instantiation of security rules) can be formally expressed as follows:

$\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall r \in R, \forall a \in \mathbf{A}, \forall v \in V, \forall c \in C,$

*Permission* (*org, r, v, a, c*) $\wedge$

*Empower* (*org, s, r*) $\wedge$

*Consider* (*org, α, a*) $\wedge$

*Use* (*org, o, v*) $\wedge$

*Hold* (*org, s, α, o, c*)

→ *Is permitted*(*s, α, o*)

This rule means:

- ***If*** a security rule specifies that "in organization *org*, role *r* can carry out the activity *a* on the view *v* when the context *c* is True",
- If "in *org*, role *r* is assigned to subject *s*",
- If "in *org*, action *α* is a part of activity *a"*,
- *If "*in *org* object *o* is part of view *v"* and,
- If "the context *c* is True for the triple (org, s, *α, o)*".
  ***Then*** the subject *s* is allowed to carry out the action *α* on the object *o.*

OrBAC is able to specify fine-grained access control policies. It provides a global security policy framework while respecting the functioning of each organization. Moreover, a first-order based logical model is associated to OrBAC. The latter provides a formal system for specifying and reasoning about permissions, prohibitions and obligations [Abou El Kalam et al. 2003]. We can thus use this model as well as first-order axioms to interrogate the security policy (e.g., to know in which conditions (context) a certain user has access to a particular resource); to detect and resolve conflicting situations (e.g., where the actions of two rules contradict each other); etc.

In our context, not only we have to specify the security rules for each CII subsystem, but also to manage secure collaboration between these subsystems and to enforce (into CIS) the different security policies by suitable security mechanisms. OrBAC is able to achieve the first point (specify the security rules for each CII subsystem), but it does not deal with the two last.

The web services technology provides some mechanisms – in particular for collaboration – that could be interesting in our work. The next sub-section presents this technology.

#### 4.3.2.2 Web services

WS are a set of technologies that provide platform-independent protocols and standards used for exchanging data between Web applications. Software applications written in various programming languages and running on various platforms can use WS to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability is made possible through the use of open standards. In a classical WS application, XML is used to describe the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing and localizing available services:

- **XML** (*Extensible Mark-up Language*) creates "common" information formats and shares both the format and the data on the Internet/intranets [W3C 2004].
- **SOAP** (*Simple Object Access Protocol*) acts as a data transport mechanism to send data between applications in one or several operating systems. SOAP specifies how to encode an HTTP header and an XML file so that a program in one computer can call a program in another computer and exhange information [W3C 2003].
- **WSDL** (*Web Services Description Language*) is an XML-based language used to describe the services that a business offers and to provide a way for individuals and other businesses to access those services electronically [W3C 2006].
- **UDDI** (*Universal Description, Discovery, and Integration*) is an XML-based registry/directory for businesses worldwide, which enables businesses to list themselves and their services on the Internet and discover each other [OASIS 2005a].

To simplify these concepts, let us establish a parallel between WS and phone calls. XML represents the conversation, SOAP describes the rules for how to call someone, UDDI is similar to the phone book, and WSDL describes what the phone call is about and how one can participate.

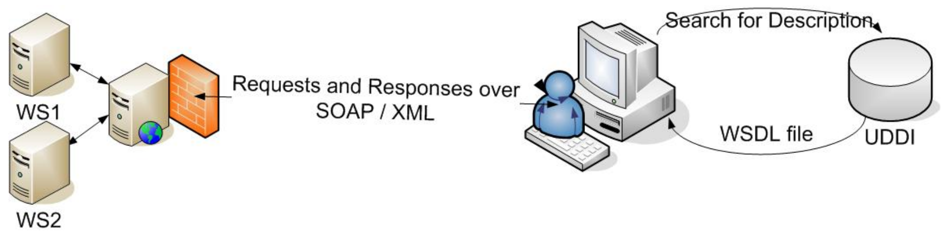

**Figure 13: Functioning of web services.**

Basically, when a user wants to use a specific WS (see Figure 13), he:

a. Contacts the UDDI to look for the WSDL file of the WS, then
b. Sends a request to the site that hosts this service, and finally
c. Receives the WSDL file containing the description of the service as well as the URL of the hosting site of the WS.

Web services (WS) have several benefits:

- **Interoperability:** WS support interoperability between different software components from different platforms.
- **Resources sharing:** WS are well-adapted to web applications where organizations share their resources.
- **Standardized mechanisms:** WS use open protocols and standards (HTTP, XML, etc.). They can be used easily with today's Web Interfaces.
- **Not restricted by firewalls:** based on HTTP protocol, web services can work through different firewalls, without specific rules changing.
- **Easiness:** the execution of a WS does not necessarily require huge resources (e.g., memory, CPU). Moreover, a small amount of code is necessary to develop a WS.
- **Compatibility with OrBAC:** it is easy to couple web services with OrBAC.

#### 4.3.2.3 PolyOrBAC

After studying the requirements of CIIs as well as existing access control models and mechanisms, we suggest using and adapting OrBAC and WS (Web Services) mechanisms to enforce secure collaboration between organizations (through mechanisms implemented into CIS). The global framework is called PolyOrBAC.

In fact, PolyOrBAC uses OrBAC to specify local access control policies (for each organization) as well as (collaboration) rules implying several organizations (see Figure 14). In this way, the same rule, for example *Permission(organization, role, activity, view, context),* could concern several internal and external accesses. Moreover, PolyOrBAC uses WS mechanisms to enforce the collaboration (see Figure 14).

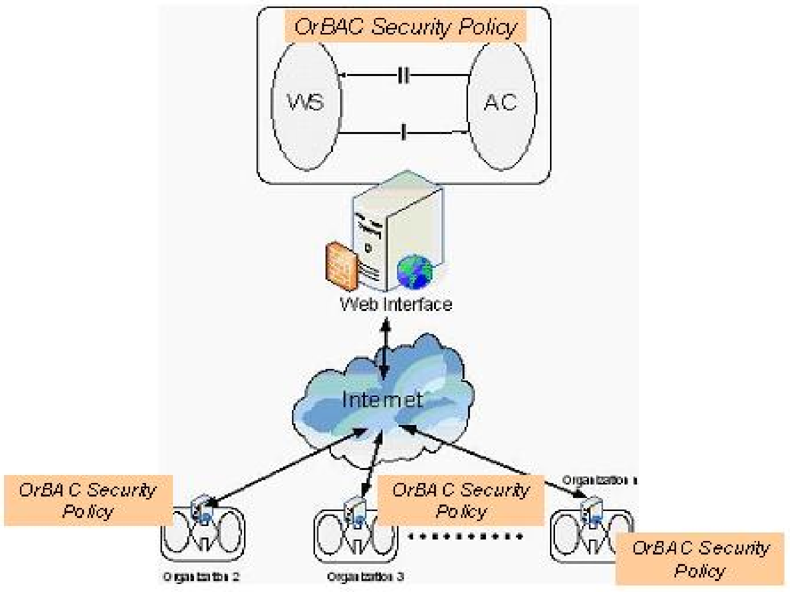

**Figure 14: The use of OrBAC for specifying local as well as remote access rules.**

Let us consider a simplified scenario. Firstly, each organization determines which resources/services it will propose to external partners. Web services are then developed on application servers, and referenced on the Web Interface to be accessible to external users. When an organization publishes its WS at the UDDI registry, the other organizations can contact it for expressing their profit-sharing. In the example below, organization *B* offers $WS_1$, and organization *A* is interested in using $WS_1$.

Secondly, organizations *A* and *B* negotiate and come to an agreement concerning the use of $WS_1$.

A and B establish then a contract and define jointly security rules concerning the access to $WS_1$.

Finally, these security rules are registered – according to an OrBAC format – in a CIS database containing the Security policy[14] (see Figure 15).

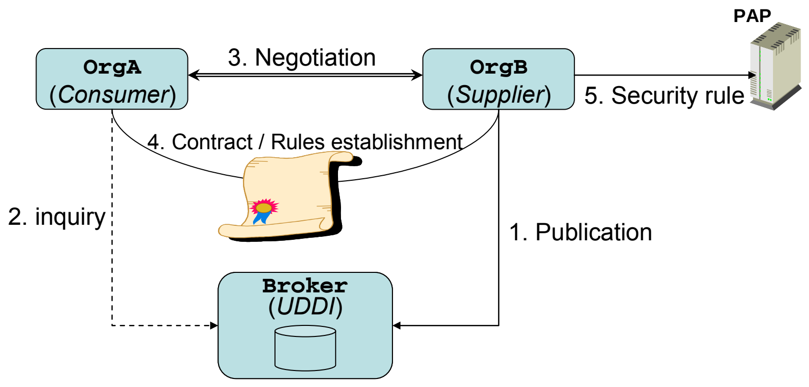

**Figure 15: Mutual negotiation of access rules for distant services.**

[14] In the OASIS/XACML (eXtensible Access Control Markup Language), this base is called a PAP (Policy Access Point).

As we use a WS-based architecture, messages exchanged (e.g., services invocations/replies) between A and B are XML files that obey SOAP protocols. These messages are controlled by both A's and B's CIS, according to the contract established between A and B.

Actually, if a user (let us note it *Alice*) wants to carry out an activity, she is first authenticated. Then, A's protection mechanisms check if the OrBAC security policy of A allows it. This activity may contain local as well as external accesses. Local accesses should be controlled according to the A's security policy, while remote accesses should respect agreement established between A and the other organizations (containing the requested services). If, for example, the Alice's Activity invokes (among others) the B's web service $WS_1$, the access to $WS_1$ should be controlled by B's CIS, according to the OrBAC security policy of B and the agreement established between A and B about $WS_1$.

PolyOrBAC offers several benefits:

- **Peer to peer approach:** we use a decentralized architecture where organizations mutually negotiate their common rules; each organization is responsible for its users authentication and is liable for their use of other organizations' services; it also controls the access to its own resources and services.
- **Independence:** even if all PolyOrBAC rules are specified according to OrBAC, organizations are loosely coupled, e.g., each organization keeps its specific security policy, security objectives, services, applications, operating system, etc.
- **Information non-disclosure:** the WS technology allows communications between organizations without intimate knowledge of each other's IT systems behind the firewall / CIS; moreover, even if remote accesses are possible, it is not necessary to know the hierarchical composition of the other organizations.
- **Extensible structure:** the OrBAC extensibility and the WS easiness and standards facilitate the management and the integration of new organizations (with their users, data, services, policy, etc.).

### 4.3.3 Perspectives

This work can be extended by studying other issues related to:

- the enforcement of obligations;
- the study of the integrity and the availability of data, functions and services; and
- the implementation of the authentication step as well as a PolyOrBAC scenario for CII (into CIS).

In fact, the access control decisions must be enforced by mechanisms for the authentication of users and for the prevention of unauthorized operations, but also for enforcing obligations.

In the context of CRUTIAL, we distinguish two kinds of obligations:

- *internal obligations*: mandatory actions that should be carried out by the system, e.g., notify the administrator, restart the server or block the access to a particular service when one or a sequence of events (e.g., an intrusion, an accidental fault) arise,
- *external obligations*: mandatory actions that should be carried out by external entities, e.g., if the energy level reaches (goes down to) a specific threshold, the operator must activate a specific regulation and control rule. Such obligations are not performed automatically by the system, but the system must monitor that they are carried out by the external entities.

Naturally, OrBAC is able to specify obligation rules. The remaining work consists in defining suitable mechanisms for implementing these rules on the heterogeneous platforms (hardware and software) composing the architecture, so as to guarantee not only the confidentiality, but also the integrity and the availability of CII's data, functions and services.

For example, when studying the availability, we should take into account several aspects such as:

- The obligation to provide sufficient resources for the fulfilment of the allowed activities by authorized users (when needed), including when some of the components fail, due to accidental events or malicious attacks. For this aim, we can call on fault tolerance mechanisms such as redundancy, distribution and diversification of servers, etc.
- The detection of processes and channels failure (at the system level).
- The definition and the enforcement of a scheduling policy. The latter should take into account the constraints (e.g., task's pre-emption) and the needs of each application.
- Etc.

Finally, concerning the integrity, we are considering extending PolyOrBAC by rules controlling the information flow to cope with applications of different criticalities, especially when users access to external organizations. In this way, accesses should be granted or denied, not only according to "classical" OrBAC rules, but also to information flow rules.

## 4.4 Monitoring and Failure Detection

### 4.4.1 Motivations

The main diagnostic problems that can be found in a modern power grid distributed infrastructure as CRUTIAL are the following:

- The use of SCADA sub-systems which were not designed to be widely distributed and remotely accessed, and that do not cover security issues (they grew-up as stand-alone systems). SCADA systems are not going to be redesigned or rebuilt, so they cannot be modified in order to comply with the information infrastructure needs.
- Some of the components could implement error detection and/or recovery techniques that work "against" the infrastructure needs, but it is necessary to coordinate all those tools in order to use them in the way the system wants.
- The use of large grained components: there are many interactions among sub-components, so it is difficult to link a single error with a well focused fault. Two kind of actions may follow: i) if the detected error is severe enough to require immediate action, then determine the set of all components that could harbour the originating fault, as well as the specific commands able to bring each component to a correct, consistent state; ii) otherwise, it is not beneficial, as soon as an error (or deviation) is observed, to immediately declare an entire component "failed" and to proceed to repairing or replacing it; it is thus better to collect streams of data about error symptoms and deviation detection, and proceed to fault assessment by observing component behaviour over time.
- The heterogeneity of the environment where diagnosis is performed: many different entities will coexist in the system, so each component needs specific diagnostic solutions (station sub-systems need to exchange values in order to perform the

secondary power control, whilst the substation web service provides visibility of selected information over internet).

- The goodness of a component could be related to the quality of the service provided, rather than on the absence of faults.
- The infrastructure is distributed by its very nature, so it is exposed to communication and coordination problems, as well as to those caused by hardware or operating system; it is not possible to manage these problems at the component level, but it is necessary to do that at application level, or, better, at middleware level. In order to provide fault tolerance to distributed component based applications, it is necessary to implement mechanisms which take into account the fault tolerance policies implemented by the different components within the system, and to add the necessary coordination support for the management of FT at application level.

### 4.4.2 The Diagnosis Framework

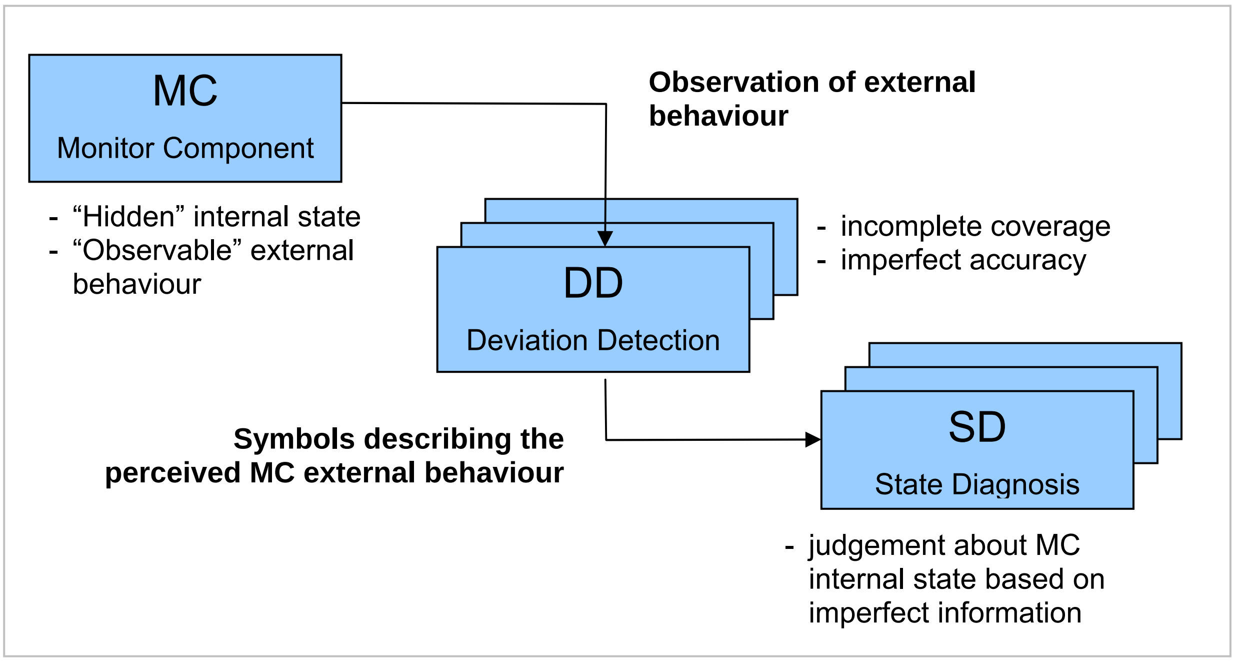
MC
Monitor Component
Observation of external behaviour
- "Hidden" internal state
- "Observable" external behaviour
DD
Deviation Detection
- incomplete coverage
- imperfect accuracy
Symbols describing the perceived MC external behaviour
SD
State Diagnosis
- judgement about MC internal state based on imperfect information

**Figure 16: The diagnosis framework.**

The diagnosis framework involves the following actors (see Figure 16):

- **Monitored Component** (MC): is the system component under diagnosis. The monitored component, when first introduced in the system, works properly; during system lifetime, the monitored component could be affected by some faults that might compromise its functional behaviour. The internal health state[15] of the component is therefore "hidden", whilst the external behaviour is "observable". Since the same observed incorrect behaviour could be caused by different faults, it is an ambiguous indicator of the internal health state of the component itself.

[15] The "internal state" is something related to the component situation with respect to faults; there is no relationship between the above "internal state" and the possible component states related with the work performed by the component itself.

- **Deviation Detection** (DD): is the entity introduced in the system in order to observe the monitored component external behaviour and to judge whether it is suitable or not. Unsuitable component behaviour could be the result of the manifestation of a fault affecting the monitored component or it could be determined by a change in the requirements of the application that is using the component monitored services.

  The deviation detection mechanism has indeed incomplete coverage and imperfect accuracy, so it can raise false positives (when it detects an inexistent deviation) and false negatives (when it does not detect an existent deviation). In critical systems both false positives and false negatives are undesired: false positives led to an early depletion of system resources, while false negatives drastically decrease the system dependability.

- **State Diagnosis** (SD): basing on information coming from the deviation detection mechanism, the state diagnosis mechanism has to guess the internal state of the component. Deviation detection information is an imperfect judgement describing the instantaneous external behaviour of the monitored component; the state diagnosis mechanism has to trace the monitored component deviations over time in order to decide whether the monitored component services continue to be beneficial or not for the rest of the system, deviations notwithstanding.

The diagnosis framework requires two information flows:

- MC ↔ DD: the deviation detection mechanism has to observe the monitored component;
- DD ↔ SD: the state diagnosis mechanism has to collect deviation detections performed by the deviation detection mechanism.

Each of the above information flows could be managed following a proactive or a reactive schema: in the proactive schema, the entity interested in fresh information has to ask for them, whilst in the reactive schema the entity that generates information has to send it to the entity interested in it.

In [Romano et al. 2002] three interaction patterns are defined in order to manage the DD ↔ SD information flow; the main differences between the alternative solutions are the granularity and frequency of the interactions:

- Continuous Monitoring: there is an interaction every time the DD performs the detection;
- Buffered Asynchronous Monitoring: to decrease communication overhead, data about deviation detections are buffered by the DD and sent to the SD in chunks;
- Failure Triggered Synchronous Monitoring: the idea is to send information only when deviations are detected, so if no deviations are detected, no interaction exists between DD and SD; when a deviation occurs, DD alerts SD and the interaction becomes continuous until SD judges that it is necessary.

The above solutions have different balances in terms of QoS vs. cost of the data fed to the SD mechanism: continuous monitoring is very costly and probably too aggressive in terms of the overhead it induces on the system; buffered asynchronous monitoring is cheaper then the continuous monitoring for the interaction cost, but requires storing capabilities in the DD mechanism and negatively affect the promptness of the SD; the failure triggered synchronous monitoring combines the advantages of the continuous monitoring (timeliness of the input data and no need for storage) and of the buffered asynchronous monitoring (reduction in communication cost), but can be impaired by omission faults in the DD.

Diagnosis activity has to be performed at different granularity levels (Fault Replacement Unit), depending on the controllability of the monitored component (e.g., when dealing with COTS and legacy subsystems) and on the cost/efficacy ratio of the detection/diagnosis/

reconfiguration operations. On one side, fine grained diagnosis is very helpful since it allows replacement of smaller parts of the system, avoiding wasting still useful subparts of the components under diagnosis. However, fine grained diagnosis incurs in higher costs from the point of view of setting up diagnosis activities. Opposite trends are instead shown by a coarse grained approach.

When diagnosis needs to be performed in a large system it is not practical to have a centralized SD entity that has to gather and analyze all the deviation detections in order to diagnose the system; this kind of centralized state diagnosis should be ultra-reliable and communication links to all the parts of the system should be guaranteed. Therefore, methods for distributed diagnosis are mandatory, where each system node decides independently about the system (e.g. which are the healthy nodes and which the faulty ones).

Considering a distributed system comprised by completely connected nodes, the Hybrid Fault-Effect Model [Walter et al. 1997] can be assumed, so that all fault classification is based on a local classification of fault-effects (to the extent permitted by the deviation detection mechanism of the node itself) and on a global classification, thus developing a global opinion on the fault-effect. Diagnosis is thus performed using a two-phase approach on a concurrent, on-line and continual basis:

1. Local detection and diagnosis, based on the node's local perception of other nodes fault status;
2. Global information collection and global diagnosis, obtained through exchange of local diagnosis.

Since each node may have a different perception of the errors created by other nodes, each node has some private values (the results of local diagnosis on remote nodes) and the goal is to ensure consistent information exchange and agreement against Byzantine behaviour. An agreement (or consensus) algorithm is thus needed in order to solve the problem.

A consensus diagnosis algorithm [Walter et al. 1997] ensures the following properties:

- Agreement: if node A and B are non-faulty, then they agree on the value ascribed to any other node.
- Validity: if node A and B are non-faulty, then the value ascribed to A by B is indeed the private value of A

In the general case, the necessary conditions to achieve consensus in spite of up to *f* arbitrarily faulty nodes are:

- at least *3f+1* nodes in the system;
- at least *f+1* rounds of message exchange;

Under the assumption of authenticated messages, which can be copied and forwarded but not altered without detection, the condition on the minimal number of nodes can be relaxed to *f+2*.

### 4.4.3 Diagnosis Solutions

The traditional diagnostic problem is the identification of failed components in a usually large set of homogeneous ones; fine grained components are the target of diagnosis, therefore, one-shot diagnosis of a collected syndrome[16] is performed.

[16] A syndrome is vector of Boolean error check outputs. deviation detection results.

Literature shows that many over-time diagnostic mechanisms are available; each approach can be mapped on the schema presented in Section  and described involving only one component and its deviation detection mechanisms. Two approaches are relevant:

- Heuristic diagnosis: heuristics are typically simple mechanisms suggested by intuitive reasoning and then validated by experiments or modelling.

  The state of the art in the panorama of heuristic diagnosis solution is alpha-count [Bondavalli et al. 2000]; this heuristic was designed in order to solve a fault discrimination problem, namely to discriminate whether the monitored component is affected by a transient fault (the component is healthy but is temporally behaving bad) or by a permanent fault (the component is physically damaged and need maintenance).

  The idea behind the alpha-count heuristic consist in counting error signals collected over time, raising an alarm when the counter passes a predefine threshold. When non-error signals are collected, the counter is decreased using a decreasing factor.

  Suppose *J(t)* is a Boolean error signal coming from the deviation detection mechanism at time *t* ("0" means no-error detected, "1" means error detected) and suppose that *K* is the internal parameter that decreases the counter value when a "no-error" signal is collected; the alpha counter $\alpha(t)$ is formally defined as follow:

$$\alpha(0) = 0$$
$$\alpha(t) = \begin{cases} \alpha(t-1) \cdot K & J(t) = 0 \\ \alpha(t-1) + 1 & J(t) = 1 \end{cases}$$

  Every time the counter is evaluated, it is also compared with a predefined threshold value $\alpha_t$ in order to discriminate if an alarm has to be raised ($\alpha(t) \geq \alpha_t$) or not ($\alpha(t) < \alpha_t$).

  Extended analyses about parameter tuning are available in [Bondavalli et al. 2000].

- Probabilistic diagnosis: probabilistic diagnosis mechanisms ([Daidone et al. 2006], [Pizza et al. 1998]) are tailored to evaluate the probabilities of the monitored component being in each of the "internal" state envisioned in the fault model, basing on symbols coming from the deviation detection mechanism. Both probabilistic diagnostic mechanisms compute a state occupancy probability vector *f(t)* at time *t*, basing both on the symbols coming from the deviation detection mechanism at time *t* and the state occupancy probability vector *f(t-1)* at time *t-1*.

  The idea behind probabilistic diagnosis is to use the Bayesian inference. Suppose to have a conjecture *x* on which we are uncertain (we believe in *x* being true with probability *p(x)*); we aim to update our belief in *x* when some new, relevant evidence is observed. Both evidence and conjecture are described as events, that is, subsets of the set of al the possible outcomes of some experiment. Using the Bayes' theorem we can write that in general:

$$p(x \mid evidence) \cdot p(evidence) = p(evidence \mid x) \cdot p(x)$$

  Interpreting the left-most probability in the above equation as the "posterior" probability of conjecture *x* (taking into account the observed evidence) and the right-most probability as the "prior" probability of conjecture *x*, we can write

$$p_{posterior}(x \mid evidence) = \frac{p_{prior}(x) \cdot p(evidence \mid x)}{p(evidence)}$$

Given a set *C* of mutually exclusive conjectures such that their union has probability 1 (e.g. health states of a monitored component), the above formula allows us to update the posterior probability of conjecture *x* given some evidence (e.g. the outcome of deviation detection) using the following formula:

$$p_{posterior}(x \mid evidence) = \frac{p_{prior}(x) \cdot p(evidence \mid x)}{\sum_{cong \in C} p_{prior}(cong) \cdot p(evidence \mid cong)}$$

Both probabilistic diagnosis approaches solve the problem with the same computational cost, but diagnosis based on hidden Markov models accounts for higher modularity and relies on a richer framework to solve diagnostic problems (e.g., helps in case of incomplete information on the involved parameters).

### 4.4.4 Properties of Diagnostic Mechanisms

Diagnosis mechanism can be analyzed basing on the following properties:

- Accuracy: the capability to correctly identify the internal state of the component despite imperfect information collected from the deviation detection mechanism;
- Promptness: how quickly the diagnosis mechanism is able to identify a change in the state of the monitored component, given that the change took place.
- Completeness: it is eventually reached, because research studies demonstrate that diagnostic mechanisms are asymptotically complete.

A trade-off exists between accuracy and promptness so, in order to get better the first, the second does get worse and vice versa.

Some considerations have to be made in order to choose at design time which diagnostic approach could be the best for a specific task:

- Accuracy: the probabilistic approach was demonstrated to use the available knowledge about the system in the best way [Pizza et al. 1998], so it has the highest accuracy that can be attained basing on the given knowledge. The heuristic approach shows to be almost accurate as the probabilistic one [Daidone et al. 2006], provided that the internal parameters are tuned in the proper way[17].
- Computational cost: the heuristic approach needs only to perform trivial operations at each step, whereas the probabilistic approach needs $O(N^2)$ operations at each step[18]; this means that the probabilistic approach could be too costly when limited resources are available.

[17] Heuristic parameters can be easily tuned at design time using the HMM-based probabilistic approach (see [Daidone et al. 2006] for more details).

[18] *N* is the number of the states constituting the fault model, so usually about ten.

### 4.4.5 Diagnosis in the CRUTIAL infrastructure

The CRUTIAL architecture, as depicted in Section 3, envisions that facility gateways (CIS) are used in order to connect LANs (possibly grouped in facilities) through the WAN; this means that each CIS has two interfaces: one on the facility side, and the other on the WAN side (Figure 17). The facility environment is typically less hostile with respect to the WAN environment, where information flows on public and open communication media, but each CIS has still to manage the connection between “its” LANs and LANs behind other CIS elsewhere in the WAN.

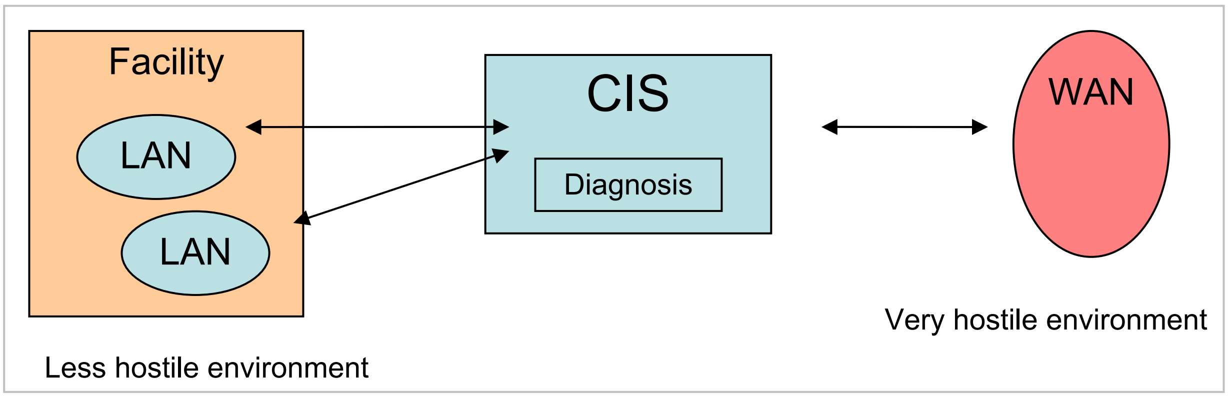

**Figure 17: CIS interfaces towards facility side and WAN side.**

Diagnosis has to be performed inside every CIS with respect to two different aspects:

- CIS self-diagnosis (diagnosis on facility side). LANs are the unit of trust or mistrust in the whole system (except for possible finer controls in specific cases), so the CIS has to monitor LAN activity at various levels (commands, information requests, …) in order to “measure” the level of trustworthiness of its LANs.

  CIS has also to diagnose itself against some faults (e.g. hardware faults, software faults, …); a wealth of diagnosis mechanisms are available to monitor and judge the correct/incorrect status of components internal to the node, in accordance with the assumed specific fault model (for example, the goal of the diagnosis could be to discriminate whether the component is affected by a transient fault or by a permanent/intermittent fault).

- CIS distributed diagnosis (diagnosis on WAN side). CIS distributed diagnosis is devoted to construct a common view about the “state” of every CIS (Figure 18); different CIS properties are eligible to be object of consensus:
  - Liveness: we want to agree whether a specific CIS is live or not (in order to distinguish if a link is down or the node itself is down);
  - Trustworthiness: we want to agree whether a specific CIS is trustable or not (it could be intruded by a hacker);
  - Consistency: we want to verify whether CIS self-diagnosis is consistent with the perception that other CIS can have about it.

  Consensus can be achieved for each property using different policies. Suppose that every CIS estimates a value in order to quantify one of the above properties: values can be first locally estimated (local step) and then exchanged with the other CIS in order to consolidate them (global step); otherwise values can be first locally estimated and judged as good or bad and then binary accusations are globally agreed upon.

Since algorithms devoted to solve consensus are costly and the cost depends on the number of involved nodes, some logical hierarchy could be introduced at run time in order to be able to scale the problem!

Many error/deviation detection mechanisms have to be available in various architectural levels in order to collect data useful to state the above properties: clock synchronisation errors, link errors, inconsistent behaviour, majority voting discrepancies (including omissions due to timing errors resulting from deadline violations), ….

Sufficiently accurate diagnosis is the prerequisite to trigger reconfiguration actions, which primarily attempt to recover the node by local actions, but in general may involve operations spanning a cluster of nodes. In fact, applying reconfiguration operations only locally to single nodes, without any coordination with other nodes somehow connected with them, could result not only in ineffective reconfiguration, but even in dangerous actions for the system. For example, consider the case when the recovery action in a node involves some reassignment of a higher load to neighbour nodes: if these, in turn, are affected by some local problem, the failure of the reconfiguration could spread in a domino effect, possibly leading to a black-out involving large part of the system. In general, each time the reconfiguration local to a node involves other nodes in the system, taking into account the health status of these last is required.

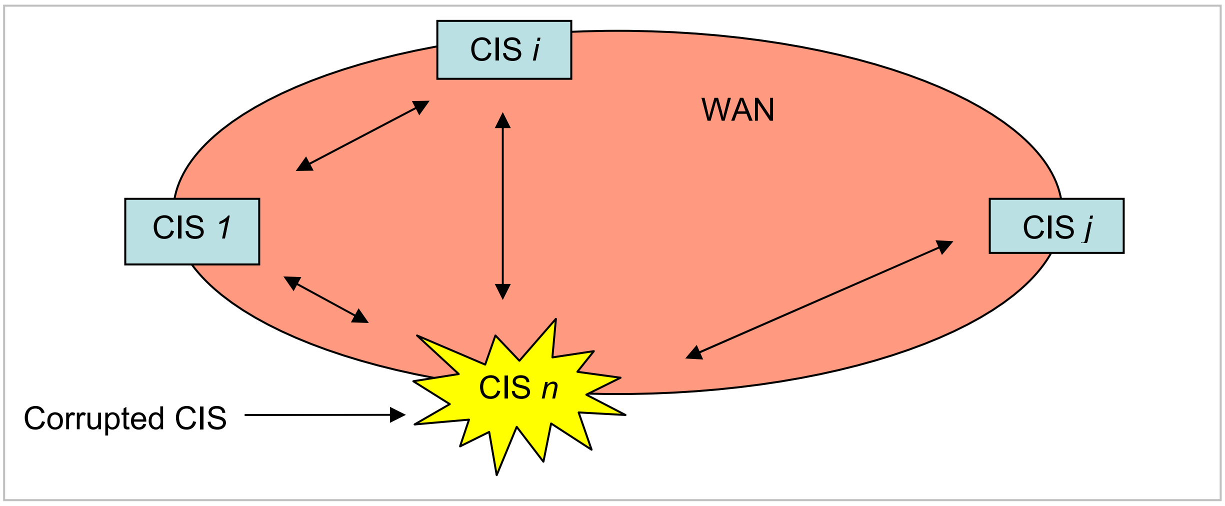

**Figure 18: CIS relationships in the WAN environment**

## 4.5 Runtime Support

The Runtime Support includes the typical operating system's services (e.g., process execution, inter-process communication), and also a set of trusted services/functions: proactive recovery, reactive recovery, and diversity management. The goal of the trusted services is to increase the overall resilience of the various system components. The following sections describe our initial ideas in terms of the necessary trusted services, but we expect that these ideas may evolve and complementary services may appear until the end of the project.

### 4.5.1 Proactive Recovery

Some system components need to be intrusion-tolerant (e.g., the Crutial Information Switches described in Section 3.2.2) and thus will use the Byzantine fault-tolerant primitives provided by the CS module (see Section 4.2). However, a Byzantine fault-tolerant algorithm typically assumes a bound on the maximum number of nodes that may be failed during its execution. In order to ensure that the system components remain intrusion-tolerant during their entire lifetime, it is important to guarantee that the assumed number of faults is never exceeded. In other words, system components that need to be intrusion-tolerant, should also be *exhaustion-safe*. Exhaustion-safety was introduced recently [Sousa et al. 2005] as a new dependability predicate that should be considered during the design of an intrusion-tolerant distributed system. A *node-exhaustion-safe* intrusion-tolerant distributed system is a system that assuredly does not suffer more than the assumed number of node failures.

One of the most interesting approaches to avoid node exhaustion due to accidental or malicious corruption is through *proactive recovery* [Ostrovsky & Yung 1991], which can be seen as a form of dynamic redundancy [Siewiorek & Swarz 1992]. The aim of this mechanism is conceptually simple – components are periodically rejuvenated to remove the effects of malicious attacks/faults. If the rejuvenation is performed frequently often, then an adversary is unable to corrupt enough resources to break the system. Proactive recovery has been suggested in several contexts. For instance, it can be used to refresh cryptographic keys in order to prevent the disclosure of too many secrets [Herzberg et al. 1995][Herzberg et al. 1997][Garay et al. 2000][Zhou et al. 2005][Cachin et al. 2002][Zhou et al. 2002b][Marsh & Schneider 2004]. It may also be utilized to restore the system code from a secure source to eliminate potential transformations carried out by an adversary [Ostrovsky & Yung 1991][Castro & Liskov 2002]. Moreover, it may encompass the substitution of software components to remove vulnerabilities existent in previous versions (e.g., software bugs that could crash the system or errors exploitable by outside attackers). Vulnerability removal can also be done through address space randomization [Bhatkar et al. 2003][Bhatkar et al. 2005][Forrest et al. 1997][Xu et al. 2003], which could be used to periodically randomize the memory location of all code and data objects. This technique is explained in Section 4.5.3.

Proactive recovery seems appropriate to build node-exhaustion-safe intrusion tolerant distributed systems. Intuitively, by using a well-planned strategy of proactive recovery, nodes may be rejuvenated sufficiently often in order that the assumed number of node failures is never violated. However, care should be taken when choosing the adequate system model and architecture to deploy a proactively recovered system. For instance, it is not possible to achieve node-exhaustion-safety if the system is designed under the asynchronous model with asynchronous proactive recovery. The simple task of timely triggering a periodic recovery procedure is impossible to attain under the pure asynchronous model, namely if it is subject to malicious faults. For a detailed discussion on this topic, see [Sousa et al. 2005].

Proactive recovery is useful to periodically rejuvenate components and remove the effects of malicious attacks/failures, as long as it has timeliness guarantees. In fact, the rest of the system may even be completely asynchronous – only the proactive recovery mechanism needs synchronous execution. This type of requirement made us believe that one of the possible approaches to use proactive recovery in an effective way, is to model and architect it under a hybrid distributed system model [Verissimo 2006].

In this context, we make use of the Proactive Resilience Model (*PRM*) – a more resilient approach to proactive recovery based on the *Wormholes distributed system model* [Verissimo 2006]. The *PRM* defines a system enhanced with proactive recovery through a model composed of two parts: the proactive recovery subsystem and the payload system, the latter being proactively recovered by the former. Each of these two parts obeys different timing assumptions and different fault models, and should be designed accordingly.

The payload system executes the “normal” applications and protocols. Thus, the payload synchrony and fault model entirely depend on the applications/protocols executing in this part of the system. For instance, the payload may operate in an asynchronous Byzantine environment.

The proactive recovery subsystem executes the proactive recovery protocols that rejuvenate the applications/protocols running in the payload part. This subsystem is more demanding in terms of timing and fault assumptions, and it is modelled as an abstract distributed component called *Proactive Recovery Wormhole* (PRW). By abstract we mean that this component admits different instantiations. Typically, a specific instantiation is chosen according to the concrete application/protocol that needs to be proactively recovered.

The architecture of a system with a PRW is suggested in Figure 19; there is a local module in every host, called the *local PRW*. Depending on the instantiation, these modules may or may not be interconnected by a *control network*. This set up of local PRWs optionally interconnected by the control network is collectively called *the* PRW. The PRW is used to execute proactive recovery procedures of protocols/applications running between participants in the hosts concerned, on any usual distributed system architecture (e.g., the Internet).

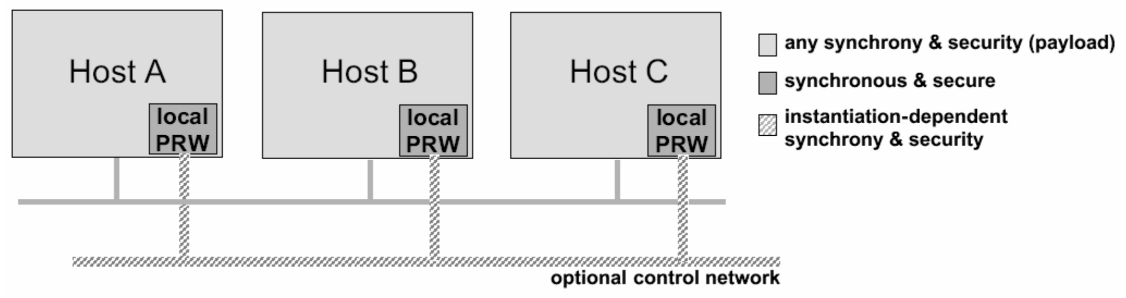

**Figure 19: The architecture of a system with a PRW.**

Conceptually, a local PRW is a module inside a host and separated from the OS. In practice, this conceptual separation between the local PRW and the OS can be achieved in either of two ways: (1) the local PRW can be implemented in a separate, tamper-proof hardware module (e.g., PC appliance board) and so the separation is physical; (2) the local PRW can be implemented on the native hardware, with a virtual separation and shielding between the local PRW and the OS processes implemented in software.

The local PRWs are assumed to be fail-silent (they fail by crashing), and local processing is synchronous, i.e., there exists a known upper bound on the local processing delays. As mentioned, a PRW instantiation may or may not have a control network. For instance, if a proactive recovery procedure only requires local information, then the control network is expendable. Even when the control network is required, its characteristics will depend on the specific requirements of the proactive recovery procedure.

The PRW has the ability to periodically execute well-defined functions in known bounded time through a *periodic timely execution service*. This service is defined as follows:

- Given any function $F$, with a calculated worst case execution time of $T_{Xmax}$, an execution interval $T_D$, and a time interval (period) $T_P$, satisfying $T_{Xmax} < T_D < T_P$, then $F$ is triggered by the PRW periodic timely execution service at real time instants $t_i$ (the $i$-th triggering occurs at instant $t_i$), with $T_D < t_i - t_{i-1} \leq T_P$, and $F$ terminates within $T_D$ from $t_i$, $\forall i$.

A system using a PRW can be made node-exhaustion-safe under certain conditions. The idea is that if it is possible to lower-bound the exhaustion time (i.e., the time needed to produce *f+1* node failures) of every system execution by a known constant $T_{exh}$, then node-exhaustion-safety is achieved by assuring that $T_P + T_D < T_{exh}$. Given that at most *f* faults are produced during any two consecutive rejuvenations, it is guaranteed that no more than *f* faults will ever be produced at the same time during the entire execution of the system. A formal proof of this reasoning is presented in [Sousa et al. 2006].

### 4.5.2 Reactive Recovery

As described in the previous section, proactive recovery is a crucial service if one wants to build intrusion-tolerant system components that are simultaneously exhaustion-safe. Reactive recovery can be seen as a complementary approach to proactive recovery, in the sense that it may trigger recoveries sooner when malicious behaviour is detected. These early recoveries may have benefits not only in terms of performance, but also in terms of system safety. Proactive recovery guarantees exhaustion-safety as long as recoveries are faster than fault production, i.e., if recoveries take less time than a lower-bound on the time needed to produce *f+1* node failures. This lower-bound is calculated at design time and must have a very high coverage [Powell 1995]. However, during system execution, malicious adversaries may prove to be more fierce than expected and may have the ability to compromise *f+1* nodes within the interval between two consecutive recoveries. Proactive recovery alone is not sufficient to maintain exhaustion-safety in this scenario (because design time assumptions were violated), but reactive recovery has the ability to defend the system against such fierce attacks if it is possible to detect the malicious behaviour of some nodes before *f+1* being compromised.

### 4.5.3 Diversity Management

The different nodes of a distributed intrusion-tolerant system (e.g., the Crutial Information Switches described in Section 3.2.2) need to be different – or diverse – in order to have a different set of vulnerabilities. Otherwise, an attack that is effective against one node is effective against all of them, and a coordinated attack could compromise all the nodes almost at the same time.

Many different diversity techniques have been proposed in the past targeting accidental and/or malicious faults. For instance, design diversity [Randell 1975] and N-version programming [Avizienis & Chen 1997] consider only accidental faults. On the other hand, [Joseph & Avizienis 1988] is a seminal work on using diversity to improve security, and, more recently, [Littlewood & Strigini 2004] presented an important study on diversity in the security domain.

[Obelheiro et al. 2006] identify several possible axes of diversity, i.e., several components of a system that may admit different instances: application software, administrative domain, physical location, operating system, and hardware. In the next sections we describe each of these axis, and it is discussed how they can be used to increase the diversity of the CRUTIAL system components.

#### 4.5.3.1 Application Software

Diversity of application software is the most usual manifestation of design diversity. The reason is that application software is often the only component over which an organization has total control, possibly having access to its specification and design details. This type of diversity can be obtained using the previously mentioned N-version programming methodology [Avizienis & Chen 1997]. The techniques suggested by Forrest et al. [Forrest et al. 1997] are also mostly to be used on applications: reordering of application code; reorganization of the memory layout of applications; modifications to process initialization.

A drawback of application implementation diversity is the cost of implementing different variants of software. However, this cost does not grow in a linear fashion: studies have shown that implementing a variant costs around 70–80% of the cost of the initial variant [Deswarte et al. 1998]. This reduction is expected, since requirements elicitation and analysis, a costly phase of the software development process, can be reused for all variants. The black-box tests of the application can be used for all variants as well. There have been several studies on N-version programming, including a (somewhat controversial) study that concluded that there is some correlation among bugs found in different variants, so failure independence is not necessarily guaranteed by this technique [Knight & Leveson 1986].

In the context of CRUTIAL, it would be interesting to use N-version programming in the design of intrusion-tolerant system components. However, we predict that the costs associated would be prohibitively high in order to adopt this design/implementation strategy in CRUTIAL.

#### 4.5.3.2 Administrative Domain and Physical Location

It is a well-known fact that many security compromises are perpetrated through social engineering [Winkler & Dealy 1995]. Distributing the components of an intrusion-tolerant service across several administrative domains seeks to reduce that problem and to hinder the use of social engineering by an intruder. The idea of administrative domain diversity is to put different systems under the responsibility of different administrators, which may apply different security management policies. Such policies encompass several issues: software used for protecting the local security domain, configuration and placement of firewalls and intrusion detection systems, as well as how these domains are organized and which security policies apply to their users. The importance of having different administrators was already pointed out for systems based on Fragmentation-Redundancy-Scattering in [Deswarte et al. 1991].

Location diversity consists of placing several physical components of a system in different sites. That distribution is an important defence against physical threats, either of a malicious nature (theft or destruction of hardware or electrical infrastructure, for instance) or an accidental one (the so-called “acts of God”, such as floods, earthquakes, fires). Natural disasters are, in the vast majority of cases, isolated events that ensure failure independence. On the other hand, systems that are so important that coordinated attacks against several installation sites must be considered, require stringent physical security policies.

It is evident that maintaining a number of adequate physical facilities to accommodate computing systems has a cost. However, often an organization that has multiple administrative units already has such facilities, or it can adapt existing facilities without excessive expenses. Furthermore, location diversity can be combined in a synergetic manner with administrative diversity. Components located in different places and administered by different people tend to present greater failure independence, and the conjugation of these measures helps to rationalize the costs of adopting these two axes of diversity.

In the context of CRUTIAL, different administrative domains (and physical locations) will exist as a natural consequence of the WAN-of-LANs architecture described in Section 3.1. However, currently, it is not clear if intrusion-tolerant services will be deployed across domains (i.e., across LANs) or only within domains. This is one of the items in our current research agenda.

#### 4.5.3.3 Operating System

The operating system (OS) plays a fundamental role in the security of a system. It controls all the machine's resources, which means that if the OS has a vulnerability then all the system can be vulnerable, even if the application has no design or implementation fault. The rationale is that applications deal with abstractions created and managed by the OS — files, processes, memory segments — so a vulnerability might allow an attacker to manipulate these abstractions freely. For this reason, a vulnerable OS is the Achilles' heel of a system, irrespectively of the robustness of the software running on top of it. Moreover, current OSs are large and complex and the number of design faults is believed to be proportional to this complexity [Hoglund & McGraw 2004]. Therefore, this axis of diversity is critical to guarantee the independence of machine failures.

The implementation of OS diversity is facilitated by the existence of standard APIs like IEEE POSIX (Portable Operating System Interface). A standard API make porting an application to a different OS easier, reducing costs and making viable a higher degree of diversity. However, if an API itself has a semantic vulnerability, like allowing a race condition [Bishop & Dilger 1996], it is probably exploitable in all OSs that implement this API.

OS diversity has a cost in terms of administration personnel. Each different OS requires a specialized administrator that can configure it to be as secure as possible. In fact, not having specialized personnel is probably worse that not using diversity since the overall system may become more vulnerable instead of less vulnerable (the same applies to other COTS components like DBMSs).

Some OSs provide a solution of compromise that guarantees a certain failure independence without requiring different OSs. Available evidence tells that the most common type of attacks in the Internet are buffer overflows [Cowan et al. 2000][Chen et al. 2006]. Most variations of buffer overflow attacks, including stack smashing and heap smashing, require some knowledge of the memory organization to be successful, something that is surprisingly easy in a conventional OS. However, the requirement of knowing the memory layout lead to a recent approach that makes this type of attacks harder: to randomize the organization of the memory [Forrest et al. 1997][Xu et al. 2003][Shacham et al. 2004]. This forces the attacker to tailor the attack to each target, using a brute force attack to discover where is the memory location it has to overwrite. These attacks can be time consuming and can call the attention to the ongoing attack. Memory randomization techniques are available for OpenBSD[19] and for Linux[20] [Xu et al. 2003].

A similar technique randomizes the instruction set of the processor in order to prevent attacks that try to inject binary code [Barrantes et al. 2003]. The idea is that if each machine had its own instruction set then it would be very hard for an attacker to devise code that would run in that machine. Randomized instruction set emulation (RISE) scrambles each

[19] http://www.openbsd.org/

[20] http://pax.grsecurity.net/

byte of the program code using a function parameterized with a per-program random key; before the program is executed, each instruction is descrambled using the inverse function; an attacker would have to know the key to be able to insert code that executed correctly.

In the context of CRUTIAL, memory and instruction set randomization may be used to offer OS diversity within the different system components. Additionally, proactive and reactive recovery can be used to periodically trigger the randomization mechanisms in each component. This way, one would achieve not only spatial-diversity (component A different from component B), but also temporal-diversity (component A at time $t_1$ different from component A at time $t_2$).

#### 4.5.3.4 Hardware

Traditionally, hardware redundancy has not been based on diversity probably because the failures of identical hardware components tend to be independent. However, recent hardware bugs provide evidence that this axis of diversity can be important for intrusion tolerance. Examples of hardware bugs with security implications are the F00F Pentium bug, which allows a user to block the processor [Collins 1998], and a bug in the hyper-threading of Intel processors, which allows the unauthorized disclosure of information by a user with low privileges (e.g., stealing private RSA keys used in the machine) [Percival 2005].

Besides these design faults, hardware diversity is also important for another reason. Exploits found in the Internet are almost always targeted to a specific hardware. Therefore, if an attacker uses an exploit to attack machines with different hardware architectures, the attack will probably be ineffective, or at least will do no more harm than crash the software running in those machines. For the same reason, hardware diversity can be useful against automated attacks using worms.

In the context of CRUTIAL, intrusion-tolerant system components should make use of hardware diversity whenever possible. It is not possible to offer a specific service than enhances hardware diversity given the intrinsic physical nature of hardware, but this axis of diversity should be considered during system deployment.

## 4.6 Middleware components for electricity services on overlays

This section describes the requirements for an application-related (web) service architecture which allows fast and specific resource discovery within the dynamic environment of possible services offered for power applications, by entities ranging from medium sized DG units to small manageable loads within households. Resilience to both accidental and malicious faults, and timing constraints are of utmost importance in this environment.

From several perspectives, it can be very useful to consider electrical energy control applications as an unbounded system - automation systems for which it is not possible to establish a global view at run-time. This is especially the case if one tries to include small scale power generation and consumption into this inherently distributed control system. These small scale systems are very large in number and are owned by an almost equally large group of people or companies. They also often have very variable and/or unpredictable properties. For example, in a dispersed electricity generation context not all energy producers are available all the time and their production may vary stochastically (wind, photovoltaic); neither is it known beforehand which electrical loads, storages and generators are available for a particular control application at some instance in time (e.g. for power quality mitigation, peak shaving, real-time balancing, economic optimization, etc.). In this context, it is relevant that the information infrastructure or the supporting middleware provides basic services by which involved parties can discover relevant resources, for example a DG unit able of injecting reactive current or some load within a certain distribution grid which can be shed. Next to this resource discovery, the middleware should also provide the ability to let the involved parties communicate (service purchaser or *client*, and the service provider or *server*)

### 4.6.1 Previous work

An example in which small scale producers are involved in the control of the grid is the one of a microgrid, more specifically the Autonomous Electricity Grid [Crutial_D2 2007] which is elaborated in the course of this project. Such applications need to support not only static configurations, but also modifications during the application's execution. Due to switching of generators and loads in a dispersed generation application, the components that need to communicate will for example vary in time, and hence, the logical communication topology has to follow accordingly.

To this extent, resource discovery is important, as to be able to quickly find the appropriate systems for the application at hand among the multitude of available systems. Several architectural configurations are possible for this resource discovery, such as a centralized or hierarchical indexing system, or a decentralized system in the form of an overlay network [Vanthournout et al. 2005a]. In this microgrid application the overlay network or peer-to-peer network approach is taken, due to their inherent fault tolerance (no single point of failure), scalability, and automated management. The overlay network used allows applications to query specific resources based on attributes defining the resources by exploiting its logical semantic topology. Also, the overlay topology is automatically constructed and maintained to allow dynamic node behaviour [Vanthournout et al. 2004][Vanthournout et al. 2005a]. The logical topology is based on an XML description of node functionality, which allows clustering entities with similar functionality (electricity meters, manageable loads, storage elements and generators, etc.). This is called *group* locality. Simulations show these semantic overlay networks have a small-world property, meaning the average number of hops to reach any node from any other node is small (e.g. 4 to 5 hops).

Such peer-to-peer network needs to periodically check for modifications: entities or links may appear, disappear or re-appear due to functional behaviour (no wind), due to electrical faults (short-circuits), or due to physical faults in the info'structure (controller or network breakdown). Indeed as parameters and functionality of entities change dynamically, so does the XML description describing these entities; hence, the overlay network needs to be adapted accordingly, to both retain its logical topology and to recover from errors. This ensures the *time* locality. The continuous adaptation of the overlay network and its inherent redundancy in links between nodes make it robust to random failures, and overlay network partitioning (splitting) becomes quite improbable.

The main functionality this peer-to-peer network offers  is automatic resource discovery and the obviously related semantic routing service with attribute-based addressing. As such, a hierarchical structure is created for data and information aggregation, and for distributed cooperation and control among the entities. All entities that need to communicate within the electricity network, have to be logically connected via a peer-to-peer network which efficiently supports distributed control primitives, such as message routing, resource discovery, gossiping, etc. Gossiping (also known as epidemic algorithms) is a scalable distributed primitive for data dissemination and aggregation, based on the periodic exchange of status data by all devices with a randomly selected neighbour in the peer-to-peer network [Jelasity et al. 2005]. A low characteristic path length of the overlay network is required for efficient gossiping [Vanthournout et al. 2005a].

These semantic overlay networks fill the gap among existing decentralized resource discovery algorithms typically used in peer to peer systems [Vanthournout et al. 2005b], that is, the lack to search resources based on (a certain range of) values of multiple attributes:

- The first generation of peer-to-peer networks -such as Napster- uses centralized indexing.
- The second generation –such as Gnutella or variants such as Kazaa- uses decentralized indexing, but with a purely random overlay topology. Queries are forwarded to every single neighbour of each node receiving a query, resulting in *broadcast queries* or *flooding*. To make this flooding somewhat practical, a maximum hop limit per query is determined (maximum number of times a query is forwarded) which results in fewer messages per query, but also in a limited scope for the query.
- The newest generation (Chord, CAN, Tapestry, Pastry, etc.) of peer-to-peer protocols uses so called Distributed Hash Tables (DHT), which results in a logical network topology and deterministic searches based on a hashed key, such as an attribute value or a unique resource identifier (see Figure 20). Searches in this kind of overlay networks are very fast (*O(log (N))*, with *N* the network size) and these networks can also deal with some dynamism in the resources and nodes. A drawback of this approach is that, although the topology is deterministic, the topology is independent of node functionality. Also, range searches or searches for similar but not identical attributes are impossible when using hashed values, since the hashed values of similar but different attribute values are totally different.

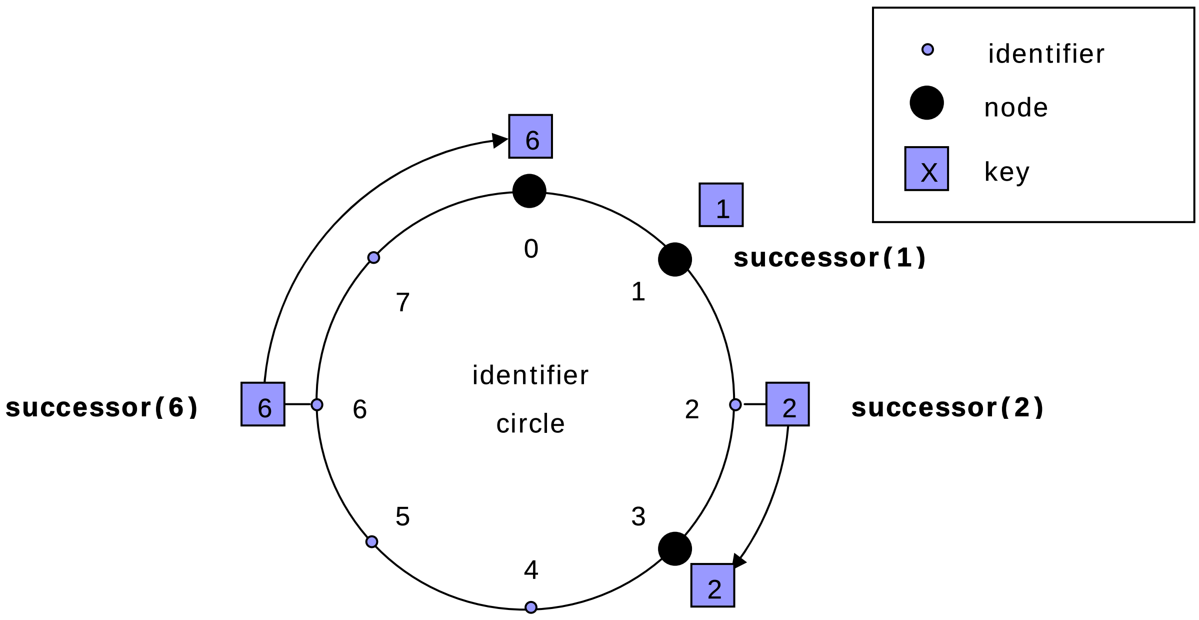

**Figure 20: DHT-ring topology, a hash maps resource key values to specific nodes with same hashed identifier value (or to the first node following this hash value in the DHT-ring in absence of an identical one). In the example the hash space is very small, namely 8 values, normally it is very large making hashed-key collisions unlikely.**

The topology of these semantic networks used in this application is based on XML descriptions of resources, where neighbours of a single node are chosen based on a distance metric between its own XML description and the other node's XML description. The smaller this distance, the larger the probability of the node becoming a direct neighbour. This topology allows to route attribute-queries based on these XML distances. Although this topology provides no deterministic query results, the overall efficiency is higher than unstructured overlay networks. Its query efficiency is lower than the deterministic overlay networks, its added value is the broader range of supported applications, thanks to the functionality based organization and the resulting support of attribute-based semantic routing. Another drawback of this semantic overlay network is the static definition of the XML-descriptions, limiting functionality of the overlay structure for the applications using the XML-attributes used in determining the semantic distances between XML-descriptions.

### 4.6.2 Generic web services for power applications

CRUTIAL will extend the earlier work on unbounded systems and overlay topologies in order to create a resilient application-related web service architecture in which electricity services function resiliently.

The aim of this service framework is to allow small scale generators (typical DG) and small, manageable loads to publish services for the power grid on a web-based platform. These services could comprise shedding of loads, production of active/reactive power, adopting some consumption profile, etc. These services can be seen as a kind of web service (also see 4.3.2.2) which can be purchased by interested parties in the power market for balancing (e.g. *virtual power plants*), power quality mitigation, fine grained load-shedding, etc.

The Internet is envisioned as the communication medium due to its ubiquitous availability and relatively cheap access. Although bandwidth and reliability of the Internet is increasing rapidly, it should still be considered an unreliable medium with only best-effort behaviour. Also other communications with small loads could be used to bridge the (very expensive) *'last mile'*, for example Power Line Communication (PLC) or Radio Frequency (RF) could be used as a low bandwidth, but real-time communication network, possibly in parallel with best-effort broadband Internet, giving rise to a natural *wormhole* (see 2.2.4) .

The main challenges for this framework can be summarized as follows:

- Electrical level
  - What services can be provided and by whom?
  - What is the effect of these services on power grid operation?
  - What are the risks when such services are abused?
- ICT level
  - *Resource discovery*: How are specific services (resources) retrieved in a decentralized fashion? Can multi-attribute queries be used? How are range queries dealt with (e.g. "active power at least 15kW")? Can we do keyword searches?
  - *Application level multicast*: How can specific resources be notified within an as short as possible timeframe without being too bandwidth consuming? An application for this could be a fine grained load shed of many small loads in the advent of a large distortion in the power grid.
  - *Access control:* Who or what company can purchase what kind of service (see also 4.3.2)?
  - *Timing and bandwidth:* What are the bandwidth requirements? What are the timing requirements?
  - *Security:* How can we secure the web application in the hostile, open environment the Internet is?

This framework can be seen as a typical web service platform, in which the application level is dedicated to the power grid, and in which resource discovery is done in a decentralized fashion, dealing with an environment which is much larger (millions of loads and generators) and much more dynamic (many loads are added and removed every minute, grid configuration are changed, etc.) than typical web services as seen nowadays.

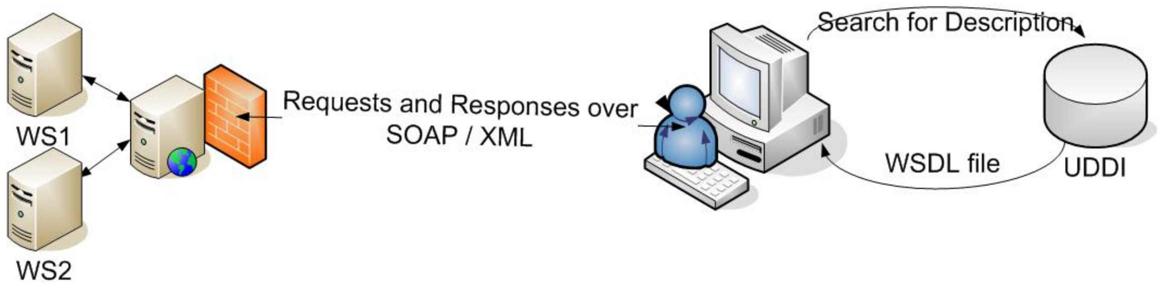

**Figure 21: a typical layout for a 'traditional' web service**

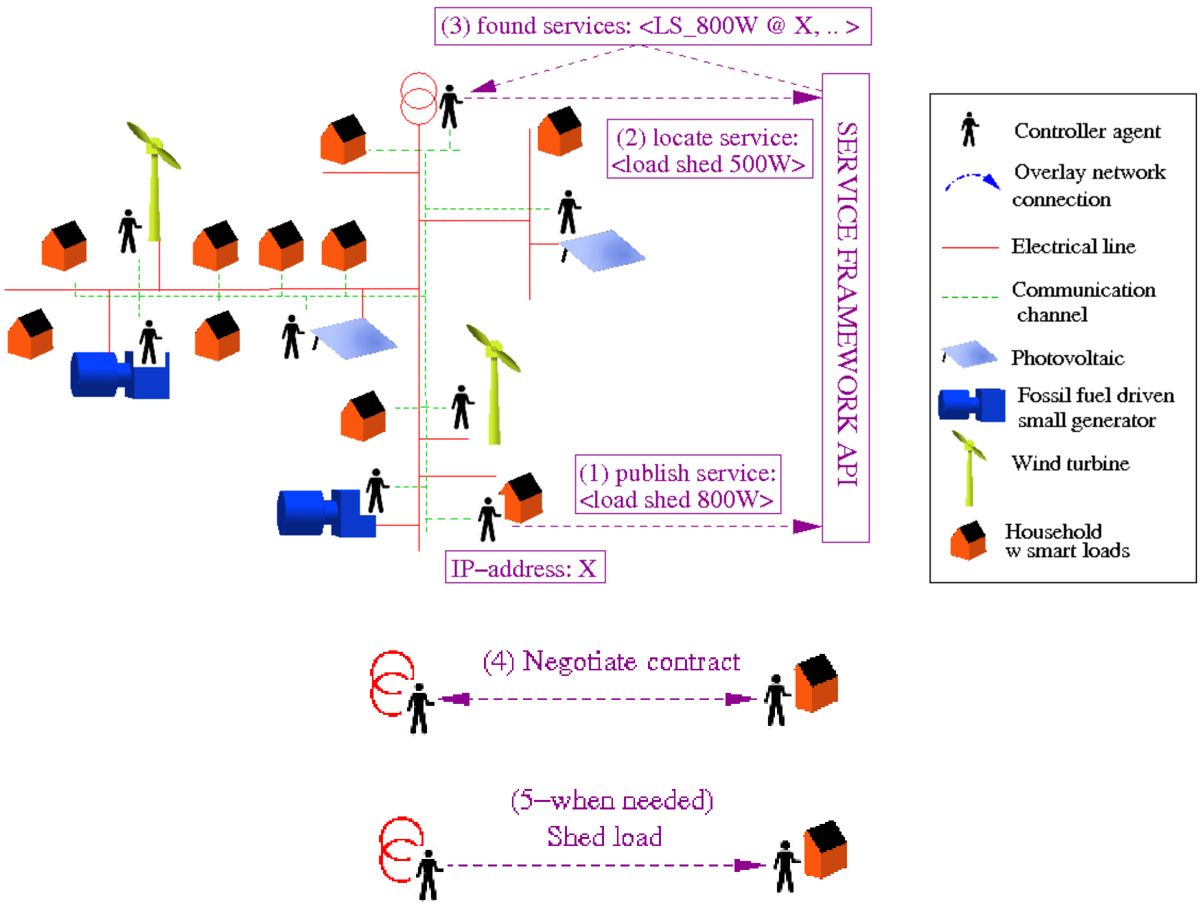
(3) found services: <LS_800W @ X, .. >
(2) locate service: <load shed 500W>
SERVICE FRAMEWORK API
(1) publish service: <load shed 800W>
IP–address: X
Controller agent
Overlay network connection
Electrical line
Communication channel
Photovoltaic
Fossil fuel driven small generator
Wind turbine
Household w smart loads
(4) Negotiate contract
(5–when needed) Shed load

**Figure 22: Example of small load publishing electrical service (load shed), and some party contracting the service and eventually using it when needed.**

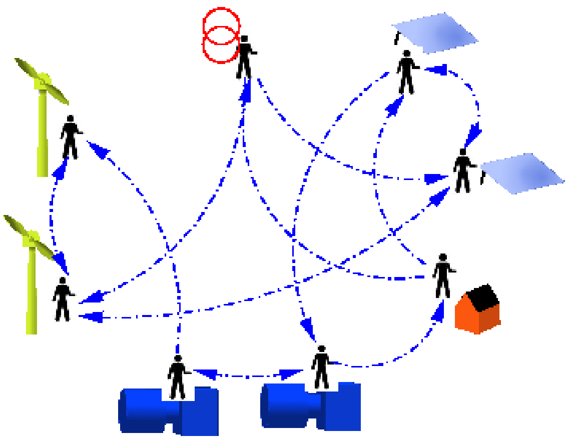

**Figure 23: Example logical overlay network used in Figure 22 to publish and discover electrical services**

Interoperability with emerging standards for data exchange in distribution networks can be envisioned (IEC 61580, "Communication networks and systems in substations"). It should be supported by flexible software modules at middleware level, one for each device connected to the network, which periodically scan their environment and put up/break or modify the connections. This provides a network abstraction to the applications allowing them to communicate based on their functionality (i.e. logical architecture), rather than on the physical topology of the network.

### 4.6.3 Resilience of overlay networks

Although overlay networks are inherently quite tolerant to accidental faults (much redundancy in overlay links and automatic reconfiguration), malicious attacks do can harm these networks a lot. Therefore, we have to look into the different known attack models on overlay networks (some are generic attacks on overlay networks, others specific for structured or unstructured networks), and see how we can make our approach more resilient against these kind of attacks. Since we plan on working with (partially?) new overlay protocols, other weaknesses may be hidden which are not described in literature today, so we have to keep an eye open for these weaknesses too. Many attack models are based on the fact that there is an unbounded, relatively anonymous number of nodes in an overlay network, which complicates access control, and all these nodes have only a limited, local view on the system, which makes detection of large scale, coordinated attacks very hard.

We give a short overview of some known attack models on peer-to-peer systems:

- *Denial of Service (DoS) attacks* are a generic kind of attack in open networks with shared bandwidth such as the IP-based Internet. A malicious users can try to inundate a part of an IP network by sending a large amount of packet, consuming all bandwidth which denies any useful communication over this part of the network. Peer-to-peer networks can be as well as any other Internet-based application the victim of this kind of attack.
- *Traffic amplification attacks* are a way to perform a DoS attack, abusing p2p network protocols, namely the flooding protocol used for resource discovery by unstructured overlay networks. The attack is performed by sending a lot of queries to different nodes in the overlay network, which forward these queries to all their neighbours, which of course tremendously amplifies the original query-messages. This attack can

be aimed at knocking out the p2p application as well as any other application sharing the same bandwidth.

- *Attacks abusing stabilization protocols* are again a way to perform a DoS attack abusing p2p protocols. In this case the attack is directed at the stabilization protocols, which serve at optimizing a structured overlay topology when nodes enter or leave the network. These protocols consume a substantial amount of bandwidth. The attack is therefore performed by quickly entering and leaving the overlay network with several nodes, and doing so, constantly invoking the stabilization protocol. The extra bandwidth consumption may eventually lead to a DoS attack on the p2p application or any other application sharing the bandwidth.
- *Bootstrapping attacks* are based on the fact that any node in a p2p network has to enter the network using a single point of entry, namely a random node already part of the network. In a bootstrapping attack, new nodes are tricked into using some malicious node as a bootstrapping node. The new node thinks he enters the p2p network, but in fact becomes part of a malicious counterpart without realizing so.
- *Application level attacks* are a broad class of attacks, and heavily depend on the application implemented on the p2p network. Typical uses of p2p are file sharing applications, and typical abuses here are the distribution of infected files (containing viruses, add-ware, etc.) over the p2p network.
- *Query falsification* returning not forwarding queries or false query results.
- *Overlay topology attack* abuse stabilization protocols in order to disturb the logical p2p topology. This may lead to more powerful (malicious) nodes, being able to falsify many queries or even to partition (split) the p2p network.

# 5 ANNEXES

## 5.1 Diagnostic mechanisms

### 5.1.1 Alpha-count

Alpha-count [Bondavalli et al. 2000] is a class of count-and-threshold mechanisms created in order to discriminate between transient faults and intermittent faults in computing systems; alpha-count is equipped with internal parameters, designed to be tuned to suit environmental variables (such as transient fault rate, intermittent fault occurrence patterns).

The first basic mechanism, the single-threshold alpha-count, gathers all signals related to the correctness of a given component, as time goes on, from any available error detector, weighing down signals as they get older, to decide the point in time when keeping a system component on-line is no longer beneficial.

A simple, albeit satisfactory, formulation of the filtering function $\alpha(t)$ is the following:

$$\alpha(0)=0$$
$$\alpha(t)=\begin{cases}\alpha(t-1)\cdot K & J(t)=0\\ \alpha(t-1)+1 & J(t)=1\end{cases}$$

where:

- $\alpha(t)$ is a score associated to the not-yet-removed component to record information about the failures experienced by that component
- $J(t)$ indicates the $t^{th}$ signal on the component ($J(t)=1$upon failure, $J(t)=0$ otherwise).

When the value of $\alpha(t)$ exceeds a given threshold $\alpha_T$, the component is diagnosed as affected by a permanent or a dangerously frequent intermittent fault; the values to be assigned to the parameters $K$ and $\alpha_T$ so that the strategy works best depend on the expected frequency of permanent, intermittent and transient faults and on the probability $c$ of correct judgements of the used error signalling mechanism. As a first flavour of the meaning of $K$ and $\alpha_T$, note that $\lceil \alpha_T \rceil$ represents the minimum number of consecutive failures sufficient to consider a component so bad to be removed, while $K$ represents the ratio by which $\alpha(t)$ is decreased after a success, so something related to how long memory of previous failures is retained.

Seeking for peculiar performances, the filtering function $\alpha(t)$ may be given formulations different from the above "multiplicative" expression; for example, the "additive" expression below associates a constant decrement $dec \geq 0$ to each errorless computation, while in the "multiplicative" expression the decrement is proportional to the current count:

$$\alpha(t)=\begin{cases}\alpha(t-1)+1 & J(t)=1\\ \alpha(t-1)-dec & J(t)=0 \quad \alpha(t-1)-dec>0\\ 0 & \alpha(t-1)-dec\leq 0\end{cases}$$

With the above "additive" function, the decay time of the current value is proportional to the value itself, while in the "multiplicative" form it depends (roughly) only on the parameter $K$.

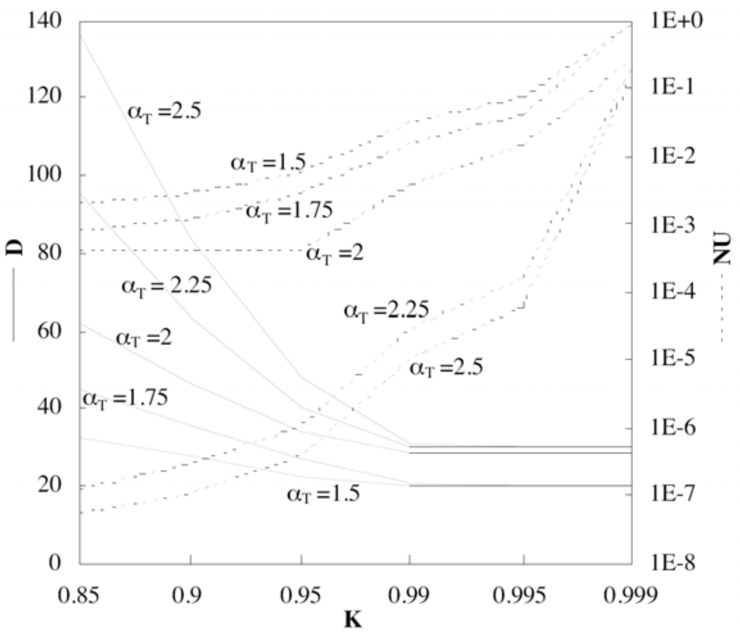

**Figure 24: Values of $D$ and $NU$ as a function of $K$ for varying $\alpha_T$.**

This would result into different optimal settings for the mechanism's parameters, and also into differences in behaviour; however, the modelling and the analysis will be worked out for the "multiplicative" form only, since it is straightforward to adapt the methods used to the "additive" form (and of course to other variants of the a function as well).

The behaviour of the single-threshold alpha-count was modelled [Bondavalli et al. 2000] by Stochastic Activity Networks (SAN), in order to evaluate 1) the average delay $D$ between an intermittent fault occurrence and its signalling, and 2) the average $NU$ of the fraction of component life[21] in which the component is healthy, but it is not effectively used because it is judged to be faulty.

Parameters $K$ and $\alpha_T$ are internal to alpha−count, i.e., under the control of the designer; their values have to be tuned, depending on the other parameters values (fault occurrences, error detection accuracy,…), in order to get the best behaviour.

Figure 24 shows the behaviour of alpha−count at varying values of $K$ and $\alpha_T$. Observe first that, while increasing values of $K$ improve $D$ (which gets lower values) and worsen $NU$, the opposite effects may be observed regarding $\alpha_T$.

At low values of $K$, $D$ markedly improves as $K$ increases, up to a region around a value $K_D$ (0.99 in case of Figure 24); for $K > K_D$ variations of $D$ become smaller and smaller. In addition, $D$ increases for increasing values of $\alpha_T$; however, it becomes less sensitive to $\alpha_T$ above the region around $K_D$.

[21] With respect to its expected life as determined by the expected occurrence of a permanent/intermittent fault.

$NU$ grows very slowly for low values of $K$ up to a region around a value $K_{NU}$, whereas it becomes worse and worse for $K > K_{NU}$. Moreover it is very sensible to the value of $\alpha_T$. Contrary to the effect $\alpha_T$ has on $D$, increasing values of $\alpha_T$ improve the figures obtained for $NU$.

If $K_D < K_{NU}$, values of $K$ in the interval $[K_D, K_{NU}]$ determine values for $D$ and $NU$ close to their optimum. It appears thus that, if no other constraints are given, values for $K$ should be chosen in such interval. However, the existence of such interval depends on other parameters as well.

Since a low $NU$ and a low $D$ are conflicting goals, the definition of the "best behaviour" for alpha−count, thus the proper tuning of its parameters, requires the definition of their relative importance. So, even in cases where the range for values to be assigned to $K$ can be identified, as in the example of Figure 24, still it has to be decided whether $D$ or $NU$ should be optimised: the former case asks for low values of $\alpha_T$ while high values optimise the latter one. More generally, tuning of parameters for a specific system can be done once the system designer has given constraints on the desired behaviour of the mechanism, e.g. "$D$ must be optimised while $NU$ must take values lower than a given threshold".

Consider now a new alpha-count scheme, where the component is kept in full service as long as its score $\alpha(t)$ is lower than a first threshold $\alpha_L$. When the value of $\alpha(t)$ grows bigger than $\alpha_L$, the component will be restricted to run application programs just for diagnostic purposes, that is, its output will only be gathered by the error signalling subsystem, and its associate alpha-count mechanism will continue to operate. In other words the component is considered "suspected", and as such its results are not delivered to the user or relied upon by the system. If $\alpha(t)$ continues to grow, it eventually crosses a second threshold $\alpha_H$, with $\alpha_H > \alpha_L$: then the component is diagnosed as affected by a permanent/intermittent fault, and the alpha-signal is issued, as in the original alpha-count, to the fault passivation subsystem. When instead $\alpha(t)$, after some wandering in the $[\alpha_L, \alpha_H]$ "limbo", becomes lower than or equal to $\alpha_L$[22], then the "suspected" component will be brought back in full service.

With this scheme, the lower threshold can be kept at a low level, to limit the detection delay, while losing not too much on the probability of throwing away good components. Actually, healthy components being hit by occasional fault bursts may be counted off into a non-utilisation period (i.e. until their alpha-count goes back under the lower threshold), but the probability of irrevocable ousting can be kept as low as desired by means of an appropriately high value for $\alpha_H$. Another adverse effect, which has to be taken into account in the dimensioning of the system, is due to the necessity of treating the errors showing up in the limbo dwellers: in fact, for continued observation of their behaviour, they need to be nursed exactly as the on-line components, putting their share of burden on the error-processing subsystem.

The behaviour of the double-threshold alpha-count was modelled [Bondavalli et al. 2000] by Stochastic Activity Networks (SAN), in order to evaluate 1) $NU$, the average of the utilisation loss of a healthy component, and 2) $F_D(d)$, the probability that the faulty

[22] In practice, a threshold value $\alpha_{L0}$, with $\alpha_{L0} < \alpha_L$, may be more appropriate here, to put a degree of hysteresis in the mechanism, with the goal of avoiding frequent "bouncing" between the "active" and "suspected" states.

component is kept in full service since $d$ steps after the permanent/intermittent fault occurrence.

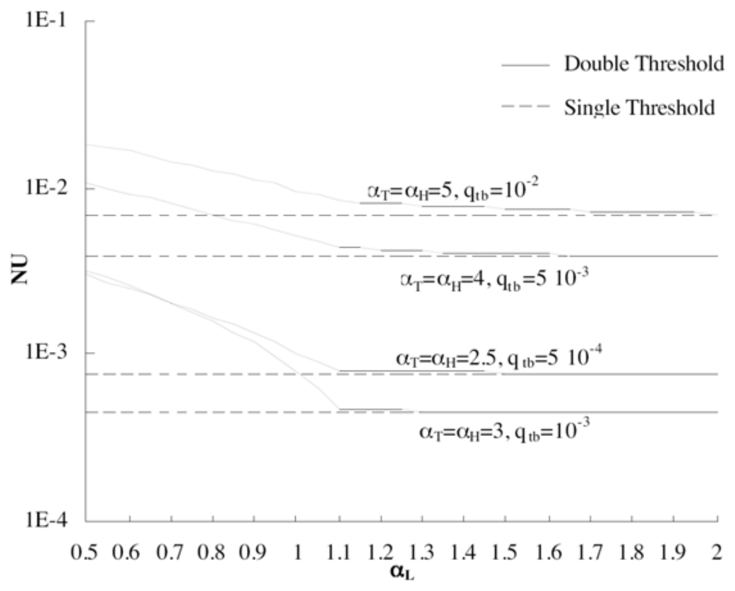

**Figure 25: Values of $NU$ against $\alpha_L$ for different values of $\alpha_H$ and $q_{tb}$, and comparison with the corresponding single-threshold case with $\alpha_T = \alpha_H$.**

The performance figures resulting from the application of the single-threshold a-count are compared with those obtainable by the double-threshold variant. In the considerations that follow, the range $[\alpha_L, \alpha_H]$ is always assumed to encompass the value $\alpha_T$. A proper setting for the parameters $\alpha_H$ and $\alpha_L$ has to be chosen. Observing that high values of $\alpha_T$ give low values of $NU$ suggests to start picking up a (probably high) value for $\alpha_T$ which fits the target $NU$, and assigning it to $\alpha_H$. Then, the value to be assigned to $\alpha_L$ has to come out as a good trade-off between a quite light negative influence on the utilisation factor of a healthy component and a satisfactorily low value for the probability to keep on-line a faulty component.

Figure 25 shows the effects of the variation of the lower threshold on the utilisation loss $NU$ of a healthy component. In this figure, four couples of plots have been depicted, each couple being formed by a plot relative to the single-threshold mechanism, and the other relative to the double-threshold case, for different values of $\alpha_H$ and $\alpha_T$. The pairs ($\alpha_T$, $q_{tb}$) have been chosen to give values of $NU$ around $10^{-3}$, considered significant for typical applications. $q_{tb}$ is the probability of occurrence of a transient fault per time unit during an abnormal period (it is higher than the probability $q_t$ of a transient fault during a normal period).

Note that, since $\alpha_T = \alpha_H$ in each plot couple, the difference between the two curves displays how much is lost of a healthy component utilisation, with regard to using the single threshold with the "best" threshold value in the range $[\alpha_L, \alpha_H]$. It is apparent that the utilisation loss is negligible in this respect with values of the lower threshold as low as 1.1. Around this value the curves exhibit a pronounced knee, which comes to no surprise, since the elemental increment in alpha-count is, in fact, 1. A first approximation choice for $\alpha_L$ is then around 1.1.

Anyway, at even lower values of $\alpha_L$ there is no dramatic loss, e.g. for $\alpha_T = 3$, (and $q_{tb} = 10^{-3}$) at $\alpha_L = 0.5$ the utilisation loss is around $4 \cdot 10^{-3}$: with such a low $\alpha_L$ setting, most permanent faults would result into the neutralisation of the affected component with virtual no delay.

Having set all internal parameters, it only remains to check if the probability to keep on-line a fault component has been lowered enough. Figure 26 shows values of the probability $F_D$ of keeping a faulty component on-line against the delay $d$ for different $\alpha_L$.

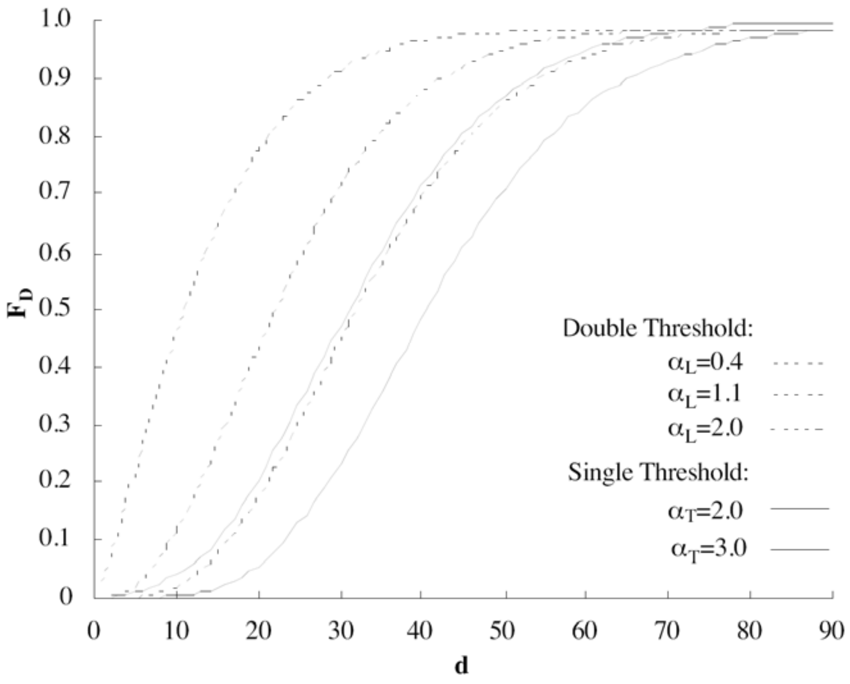

**Figure 26: Values of the probability of keeping a faulty component on-line ($F_D$) against the delay $d$ for different $\alpha_L$.**

The plot relative to $\alpha_T = 3$ allows to appreciate the improvement of the double-threshold a-count over the single-threshold with regard to the risky utilisation of a faulty component when $\alpha_T = \alpha_H$ (which is the best assignment to $\alpha_T$ with regard to the utilisation loss of a healthy component, as derived from Figure 25). As expected, the lower the value of $\alpha_L$, the higher is the improvement in the value of $F_D$. For example, in correspondence of $d = 40$ the probability of still relying upon a faulty component is about 0.55 for the single-threshold, with $\alpha_T = 3$, while that exhibited by the double-threshold with $\alpha_H = 3$ and $\alpha_L = 1.1$ is approximately 0.15. In Figure 25 for this same setting of $\alpha_H$, $\alpha_L$ and $\alpha_T$ the loss of utilisation of a healthy component is negligible; it can be concluded, in this case, that the double-threshold is clearly more appropriate in critical application contexts.

The difference between the plot relative to $\alpha_T = 2$ and that relative to $\alpha_L = 2$ represents the loss incurred by the double-threshold, in terms of $F_D$, with regard to the single-threshold enjoying the best $\alpha_T$ setting (in the range $[\alpha_L, \alpha_H]$) for $F_D$. As seen in Figure 26, while the single-threshold looks better, such difference is negligible. By the way, such a "better" behaviour regarding $F_D$ is purely speculative, as it implies worsening $NU$ .

### 5.1.2 HMM Approach

#### The HMM formalism

Hidden Markov Model (HMM) is a formalism able to represent probability distributions over sequences of observations. An HMM is basically a Markov chain whose state is not observable (the state is indeed "hidden") that emits observable symbols depending on a probabilistic function of the state. Suppose that $\Omega$ is the finite discrete set of the states and that $\Sigma$ is the finite discrete set of observable symbols; an HMM is composed of two sets of random variables:

- $\{Q_1, Q_{2,\ldots}\}$, that constitute an homogeneous, first order, discrete Markov chain, which assumes values in the set $\Omega$ and which determines state changes;
- $\{X_1, X_{2,\ldots}\}$, that constitute an identically distributed, discrete time stochastic process which assumes values in the discrete alphabet $\Sigma$ and which determines the observable symbols.

HMM is characterized by the following elements:

- $\Omega$, the set of $N$ states in the model; formally:
$$\Omega = \{\omega_1, \ldots, \omega_N\}$$
- $A$, the "state transition probability matrix". $A$ is an $N \times N$ matrix where the generic element $a_{ij}$ represents the probability to transit from state $\omega_i$ to $\omega_j$ at time $t$[23]. Formally:
$$a_{ij} = p(Q_t = \omega_j | Q_{t-1} = \omega_i)$$
It holds that
$$\forall i = 1, \ldots, N \quad \sum_{j=1}^{N} a_{ij} = 1$$
- $\vec{\pi}(1)$, the "initial state distribution vector". The $i^{th}$ element of $\vec{\pi}(1)$ is
$$\pi_i(1) = p(Q_1 = \omega_i)$$
It holds that
$$\sum_{j=1}^{N} \pi_i(1) = 1$$
- $\Sigma$, the set of $M$ distinct observable symbols (that is, the alphabet).
- $B$, the "observable symbol probability distribution matrix". $B$ is an $N \times M$ matrix containing the probabilities of emissions of observable symbols (belonging to $\Sigma$) from

[23] The value of $t$ is not important, because the chain is supposed to be time-homogeneous.

each state (belonging to $\Omega$). Let's define $b_k(\sigma)$ as the emission probability of symbol $\sigma$ at time $t$, given that the model is in state $\omega_k$:

$$b_k(\sigma) = p(X_t = \sigma | Q_t = \omega_k)$$

Note that the first three elements presented above are those characterizing a traditional Markov chain, while the last two are those specifically characterizing the HMM.

Some representative problems are described in the HMM literature:

- The evaluation problem: given a sequence $S$ of symbols observed over time, what is the probability that the HMM could emit $S$?
- The decoding problem: given a sequence $S$ of symbols observed over time, which is the most probable state sequence $P$ able to generate the observed sequence $S$? Informally speaking, it has to be found the most likely state sequence associated with the given observed sequence $S$.
- The training problem: given some sequences $S_i$ ($i = 1, \dots, n$) of observed symbols, how can we can adjust the model parameters $A$, $B$ and $\vec{\pi}(1)$ in order to maximize the probability of emitting each $S_i$? Informally speaking, how can we adapt the model parameters to the observed training sequences?

The HMM approach to diagnosis takes advantage of the so-called **forward probability**, an element used to efficiently solve the first and the third problems listed above. The forward probability is defined as follow: given a sequence of observations $S$ and a state $\omega_k$, what is the probability that the HMM is in state $\omega_k$ after having generated $S$?

Let $S$ be the sequence $o_1, \dots, o_L$ ($o_i \in \Sigma$, $i = 1, \dots, L$) (this means that from time 1 to time $L$ the model emitted the sequence $S$); the forward probability related to state $\omega_k$ at time $L$ is defined as $f_k(L) = p(X_1 = o_1, \dots, X_L = o_L | Q_L = \omega_k)$. The forward probability takes into account all the possible sequences of states, terminating with state $\omega_k$, which can have generated the observed sequence $S$; the forward probability is computed by the following recursive formula:

$$f_k(i) = \begin{cases} b_k(o_1) \cdot \pi_k(1) & i = 1 \\ b_k(o_i) \cdot \sum_{\omega \in \Omega} f_\omega(i-1) \cdot a_{\omega k} & 1 < i \le L \end{cases}$$

The computation cost of the forward probability $f_k(L)$ is $O(L \cdot N^2)$, where $L$ is the length of the sequence $S$ and $N$ is the number of the states in the model.

## How to use the formalism to solve the diagnosis problem

The diagnosis framework discussed in Section  is modelled in the following way [Daidone et al. 2006]:

- The monitored component (MC) is modelled as an HMM, as its internal state is hidden and its functional behaviour is instead observable.
  The model states are derived from the assumptions on faults and QoS requirements: in principle, a state could be associated to each assumed faulty condition and/or QoS level for the provided services, plus an additional state corresponding to the healthy

condition and/or nominal QoS delivery. It could be also appropriate to include states corresponding to combinations of faulty/QoS conditions.

The state transition probability matrix $A$ and the initial probability vector $\vec{\pi}(1)$ are derived from fault model and QoS requirements, which would allow determining the admitted state transitions and probabilities to associate to them.

The set of observable symbols $\Sigma$ and the probability distribution matrix $B$ are derived from the DD mechanism characteristics; it is assumed that the emitted symbols faithfully reveals the presence or absence of deviations; then, because of its imperfection, the DD may misinterpret them, as better detailed later on.

- The deviation detection subsystem (DD) is modelled as a “translation probabilities matrix” (called $G$); this matrix is used in order to translate the observable symbols emitted by the HMM according to the limited coverage and imperfect accuracy of the checks performed by DD on the behaviour of the MC.
  Symbols emitted by the MC model are perfectly coherent with the real presence/absence of deviations; the deviation detection subsystem DD emits symbols which belong to the same alphabet $\Sigma$, but, because of the limited coverage and completeness of the checks performed by DD subsystem, symbols emitted by the DD model may be different from those of the MC model. The translation probabilities matrix $G$ defines for each symbol emitted by the MC model the probability of translating it in each symbol in $\Sigma$.

  $G$ is an $M \times M$ matrix where each row is associated to a symbol emitted by the MC model, and each column is associated to a symbol emitted by the DD model. Let $\sigma_{in}$ be the symbol received by the DD model in a certain time instant $t$, and $\sigma_{out}$ the corresponding translated symbol; if $i$ and $j$ are respectively indexes of the row and the column associated to symbols $\sigma_{in}$ and $\sigma_{out}$, then the value $g_{ij}$ represents the probability that DD emits $\sigma_j$ at time $t$, given that MC emits $\sigma_i$:

$$g_{ij} = p\left(\sigma_{out} = \sigma_j \middle| \sigma_{in} = \sigma_i\right)$$

  It holds that

$$\forall i = 1, \ldots, N \quad \sum_{j=1}^{M} g_{ij} = 1$$

  A perfect DD would have value 1 along the matrix diagonal, and 0 in all the other positions, but this is an ideal case. In general, along the same row, several positions may contain values greater than 0.

- The state diagnosis mechanism uses the forward probability method (described above) to judge the internal state of the monitored component basing on the sequence of observable symbols emitted by the DD model. Since observed symbols are first emitted by the HMM and then translated by the “translation probabilities matrix”, an “emitted symbol probability distribution matrix” can be defined as follow:

$$\hat{B} = B \times G$$

  So, at receiving a new symbol from DD, SD incrementally determine the probability of the model being in each one of the states in $\Omega$, given that the received symbol was emitted; these probability values are stored in the vector $\vec{f}(t)$ defined as follows:

$$\vec{f}(t) = \left(f_1(t), \ldots, f_N(t)\right)$$

where $f_k(t)$ is the probability that MC be in state $\omega_k$ at time $t$, having processed the sequence of symbols received in the time interval $[1,\ldots,t]$.

The $\vec{f}(t)$ vector is defined as follow (it derives from the definition of the "forward probability mentioned above):

$$\vec{f}(t) = \begin{cases} \frac{1}{S_1} \cdot \hat{D}_{o_1} \times \vec{\pi}(1) & t = 1 \\ \frac{1}{S_t} \cdot \hat{D}_{o_t} \times A^T \times \vec{f}(t-1) & t > 1 \end{cases}$$

where $S_1$ and $S_t$ are scaling factor (necessary in order to guarantee $\vec{f}(t)$ to be a probability vector) and $\hat{D}_{o_t}$ is the diagonal matrix defined as follows ($o_t$ is the symbol emitted at time $t$, $\hat{b}_k(o_t)$ is the probability that the model could emit $o_t$ from state $\omega_k$):

$$\hat{D}_{o_t} = \begin{bmatrix} \hat{b}_1(o_t) & & 0 \\ & \ddots & \\ 0 & & \hat{b}_N(o_t) \end{bmatrix}$$

The computational cost of $\vec{f}(t)$ is of order $O(N^2)$ and does not depend on the particular time instant $t$; it requires the vector $\vec{f}(t-1)$ relative to the previous time instant, and last symbol emitted by the error detection mechanism.

Now, in order to take a decision on the state of component MC, the SD model has to be enriched with a pre-defined criterion on judging the probabilities in $\vec{f}(t)$ as good or not. A simple criterion would be to have a threshold vector $\vec{d}$ of $N$ positions, where the value in position $k$ represents the threshold for the probability of corresponding state $k$ in $\vec{f}(t)$: in case one of the thresholds is violated, the monitored component is signalled as no more beneficial to the system activities. Actually, more sophisticated procedures could be envisioned; for example, $\Omega$ could be partitioned into two parts, and threshold could be defined for each partition.

The probabilistic formalization of the diagnosis problem based on the hidden Markov model theory keeps into account all the three aspects involved in component's state diagnosis: the monitored component, the deviation detection mechanism and the state diagnosis mechanism. This modularity favours generality and widens the applicability of the method. The high accuracy of the proposed approach is demonstrable [Daidone et al. 2006] through the comparison with an existing optimal solution relying on the Bayesian inference theory. The approach is very general, as models are not bound to particular components or fault detection mechanisms, so it can be a support for studying different combinations of fault models, deviation detection mechanisms and heuristic techniques.

**The fault discrimination problem**

In this section we focus on the well known fault discrimination problem: we assume that components may be affected by permanent, intermittent or transient faults and the goal is to discriminate transient faults from intermittent and permanent ones.

We suppose that all fault-related events occur at discrete points in time; two successive points in time differ by a (constant) time unit (or step). Permanent faults occur with probability $q_p$ and, once occurred, give rise to an error at each following step. Intermittent faults occur with probability $q_i$; an intermittent fault repeatedly causes errors, after the fault occurrence, with constant probability $q$ . Transient faults occur with probability $q_t$ and last for only one step. An error detection subsystem is considered, which evaluates whether the component is behaving correctly or not. At each step, a binary signal is issued by the error detector towards the fault discriminator: 0, meaning that no error has been detected, or 1, if an error was revealed. The emitted signal is correct with a probability $c$, which accounts for the coverage of the error detection mechanism used.

In the following table the value assigned to the above parameters are shown:

| Parameter | Value |
|---|---|
| $q_t$ | $1 \cdot 10^{-5}$ |
| $q_p$ | $1 \cdot 10^{-6}$ |
| $q_i$ | $1 \cdot 10^{-6}$ |
| $q$ | 0.1 |
| $c$ | $1 - 10^{-5}$ |

From the above fault model, the following set of states is derived:

$$\Omega = \{OK, TR, PERM, INT, INT + TR\}$$

where:

- $OK$ is the fault-free state,
- $TR$ is the state representing a transient fault,
- $PERM$ is the state representing a permanent fault,
- $INT$ is the state representing an intermittent fault,
- $INT + TR$ is the state in which the component is affected at the same time by an intermittent and a transient fault.

The initial state distribution vector $\vec{\pi}(1)$ is assigned assuming that the component is fault-free when it starts working. Therefore,

$$\vec{\pi}(1) = (1,0,0,0,0)^T$$

The values of the state transition matrix $A$ are the following:

| | OK | TR | PERM | INT | INT+TR |
|---|---|---|---|---|---|
| **OK** | $1 - q_t - q_p -$ $q_i - q_t \cdot q_i$ | $q_t$ | $q_p$ | $q_i$ | $q_i \cdot q_t$ |
| **TR** | $1 - q_t - q_p -$ $q_i - q_t \cdot q_i$ | $q_t$ | $q_p$ | $q_i$ | $q_i \cdot q_t$ |
| **PERM** | 0 | 0 | 1 | 0 | 0 |
| **INT** | 0 | 0 | $q_p$ | $1 - q_p - q_t$ | $q_t$ |
| **INT+TR** | 0 | 0 | $q_p$ | $1 - q_p - q_t$ | $q_t$ |

Figure 27 shows the Markov chain that describes the states and the transitions between states derived from the fault model (it is the Markov chain that is hidden inside the model of the monitored component).

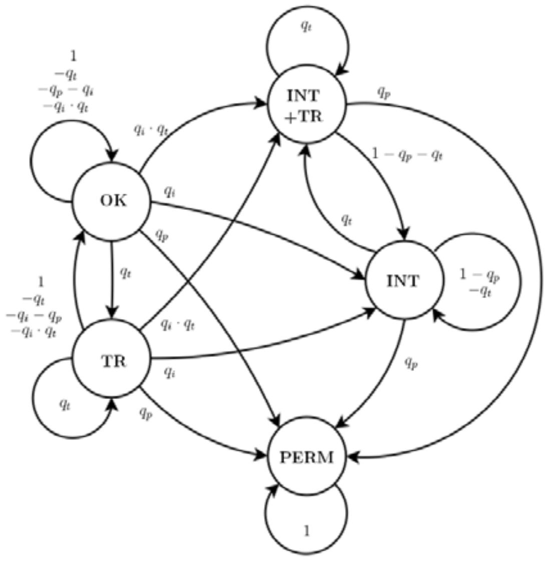

**Figure 27: The Markov chain “hidden” inside the component model.**

The set of symbols emitted by the model is $\Sigma = \{0,1\}$, where $0$ to signal the absence of errors and $1$ to signal the presence of an error. Based on the above identified symbols, the values of the observation symbol probability distribution matrix B are the following:

| | **0** | **1** |
|---:|:---:|:---:|
| **OK** | $1$ | $0$ |
| **TR** | $0$ | $1$ |
| **PERM** | $0$ | $1$ |
| **INT** | $1-q$ | $q$ |
| **INT+TR** | $0$ | $1$ |

Basing on the characteristics of the deviation detection mechanism, the following translation probabilities matrix $G$ is derived:

| | **0** | **1** |
|---|:---:|:---:|
| **0** | $c$ | $1-c$ |
| **1** | $1-c$ | $c$ |

This means that both the probability of a false negative (row “1”, column “0”) and a false positive (row “0”, column “1”) is $1-c$.

The component’s states in $\Omega$ are partitioned in two sets: the partition of states representing healthy behaviour and that of states representing faulty behaviour:

- $\Omega_{HC} = \{OK, TR\}$
- $\Omega_{FC} = \{PERM, INT, INT + TR\}$.

Then, the judgement of the diagnostic mechanism is based on a pre-defined threshold probability $P_{th}$ on the estimated probability $P(\Omega_{FC})$ for the component to be in one of the states in $\Omega_{FC}$.

The whole diagnosis framework was modelled using the Stochastic Activity Network formalism and solved through simulation[24] using the Mobius tool; Figure 28 plots the curves of the HMM diagnosis ($P(\Omega_{FC})$) for different values of the probability $q$ (intermittent fault manifestation) at increasing number of steps from the fault occurrence. The analysis was performed the in the pessimistic case: the monitored component is hit by an intermittent fault at step $0$ and $\vec{f}(t)$ has values as if the component is in a healthy state; these conditions lead to an upper bound of the steps to the identification of the component as faulty. For each threshold value $P_{th}$ on $P(\Omega_{FC})$ it can be determined the expected number of steps required by the different curves to reach such probability threshold value.

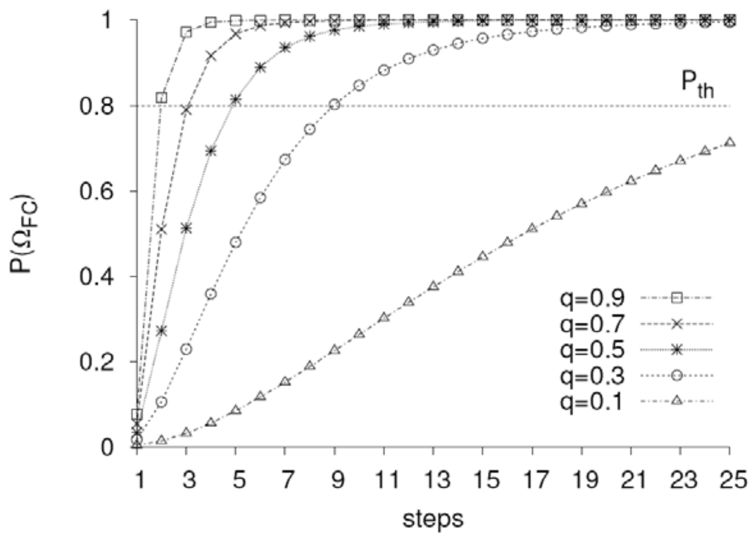

**Figure 28: Behaviour of the HMM diagnostic method on a component hit by an intermittent fault.**

Now we consider the alpha-count heuristic under the same assumptions made on the fault model and error detection mechanism presented above; simulations evaluated $P(\Omega_{FC}, \alpha(t) \geq \alpha_T)$, that is the value of $P(\Omega_{FC})$ as estimated by the HMM diagnostic mechanism at the time instant when alpha-count judges the component as permanently/intermittently faulty ($\alpha(t) > \alpha_T$).

[24] The simulation results have been obtained with a confidence level of 0.95 and a confidence interval of 0.01.

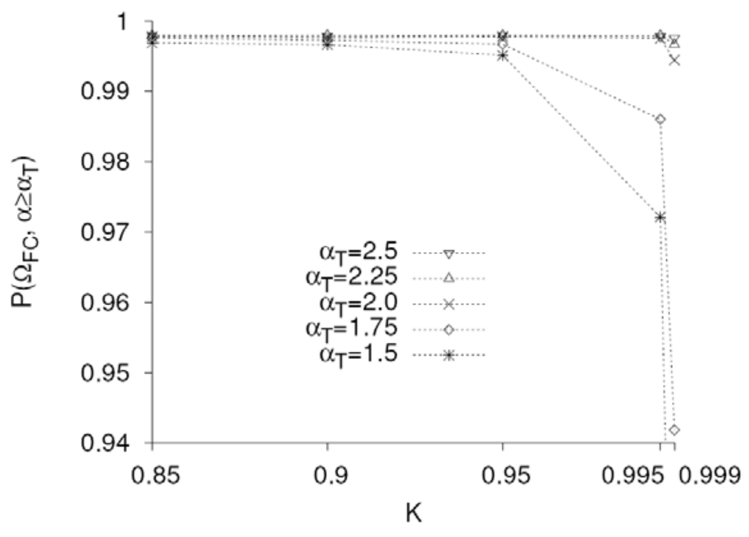

**Figure 29: Accuracy of alpha-count as determined through comparison with the HMM method.**

Figure 29 plots $P(\Omega_{FC}, \alpha(t) \geq \alpha_T)$ for different instances of alpha-count, as determined by coupling values of the heuristic parameter $K$ (on the $x$ axis), with five values of the threshold $\alpha_T$; the higher is the value of $P(\Omega_{FC}, \alpha(t) \geq \alpha_T)$, and the better is the accuracy of alpha-count. In the figure, alpha-count instances employing values of $\alpha_T \geq 2$ already show highly accurate; it is worth to note that, for higher values of $\alpha_T$, the accuracy of alpha-count is minimally influenced by the value of $K$, instead, the impact of $K$ is very relevant when $\alpha_T = 1.5$. This is not surprising, considering that the parameter $K$ sets the time window where the memory of an erroneous judgement by the error detector on the monitored component is retained: longer memory (i.e., higher $K$) accelerates the identification of components as faulty, especially when the threshold is low, and consequently also increases the probability of signalling a healthy component as faulty.

## 5.2 Access Control mechanisms

In this annex we first present a set of concepts that are used in our work on access control (i.e., a glossary related to authorization and authentication). Then, we give more details about “classical” access control models such as: Lampson, Biba, RBAC (Role-Based Access Control). Afterwards, we present the main facets of the OrBAC (Organization-Based Access Control) model. Finally, we present the XACML’s (eXtensible Access Control Markup Language) architecture.

### 5.2.1 Some Preliminary Concepts

In the rest of the section we will utilize the following concepts:

**Authentication servers** are servers that provide authentication services to users or other systems. For example, the user passes its identity and password (or certificate, smartcard, biometric data) to the authentication server; the latter verifies this data and grants the authentication proof (e.g., a credential) to the user.

**Authorization server** consults the security policy; extracts the relevant security rules; evaluates these rules with the current access parameters; eventually, invokes the conflict resolution process; and generates the corresponding credentials that permit the access to resources.

**Availability** is the readiness for correct service when needed by authorized users.

**Collaboration** refers to all processes wherein people (or machines, or applications) work together - applying both to the work of individuals as well as larger collectives and societies.

**Bell-LaPadula Model** was developed by David Elliott Bell and Len LaPadula in 1973 to formalize the U.S. Department of Defence multilevel security policy. The model is a formal state transition model that describes a set of access control rules by the use of security labels on objects, from the most sensitive to the least sensitive, and clearances for subjects. For example, a set of security labels for documents might be "Top Secret", "Secret", "Confidential", and "Unclassified".

**Biba Model** is a formal state transition system of that describes a set of access control rules designed to ensure that data are not altered. This security model is directed toward data integrity (rather than confidentiality as for the Bell LaPadula model) and is characterized by the concepts: "no write up, no read down".

**Confidentiality** is the absence of unauthorized disclosure of information or functions. It implies that information is readable only to authorized users.

**Integrity** is the absence of improper system state alterations. It implies that data is modified only by authorized users and only in an authorized manner.

**Interoperability** is the ability of products, systems, or business processes to work together to accomplish a common task.

**Security** studies problems, methods and solutions related to the three security properties: availability, confidentiality, and integrity.

**Security policy** is *the set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information*. Basically, a security policy is specified through: the security *objectives* that must be satisfied (expressed in terms of confidentiality, integrity and availability of data or metadata) and the *security rules* expressing how the system may evolve in a secure way (who has access to what and in which conditions).

**Security model** rigorously defines a security policy. Generally, a security model is a “formal system” used to specify and reason on the security policy (i.e., it is used as a basis for formal specification proofs). It is thus intended to abstract the security policy and to handle its complexity: the security model is used to represent the secure states of a system as well as the way in which the system may evolve; to verify the consistency of the security policy; to detect and resolve possible conflicts.

**Security mechanisms** are techniques used to implement the authentication and authorization, e.g., credentials, capacities, cryptographic transformations such as signature and encryption, access control lists (ACL), etc.

## 5.2.2 Examples of Classical Access Control Models

### 5.2.2.1 The Lampson model

The main entities of the Lampson model [Lamspon 1971] are subjects, objects and actions. Intuitively:

- *subjects* are active entities, e.g., users, processes;
- *objects* are passive (i.e., non-active) entities, e.g., files, directories, printers;
- *actions* enable subject to have direct access to objects, e.g., read, write, execute.

In this model, the security policy is limited to the specification of permissions, i.e. relationships between subjects, objects and actions. An access control matrix generally represents these relationships. If *s* is a subject and *o* is an object then *M*(*s*; *o*) represents the set of actions that subject *s* is permitted to execute on object *o* (see Figure 30).

In the Lampson model, it is necessary to enumerate, through a matrix of permissions, all the triples (s,o,*α*) that correspond to a permission granted by the security policy. When new subjects, new objects or new actions are introduced in the system, it is necessary to update the security policy in order to record the permissions granted to these new entities.

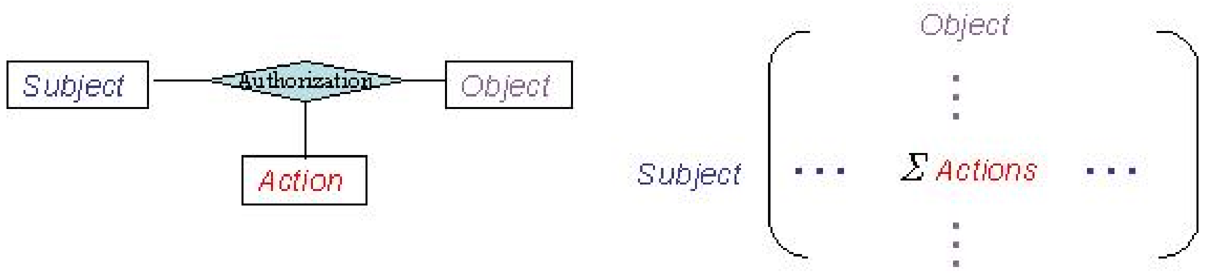

**Figure 30: The Lampson model.**

As a result, the updating of security policies specified according to this model become quite complicated. Moreover, this model is limited to permissions. Prohibitions and obligations are not supported.

#### 5.2.2.2 The Biba model

In Discretionary access control policies and models (e.g. Lampson, HRU) users can spread the rights on their data to other users (e.g., in Linux, Windows). For example, in a user *A* trusts a user *B* and not *C*, he can grant access to *B* on its (sensitive) data. *B* can copy this data, create a file *f*, put this (sensitive) data in *f*, and grant access on *f* to *C*. In this case, *C* can read *f*, even if this data is initially forbidden for him. This is an example of information leakage, the main weakness in discretionary access control.

To prevent this kind of problems, the American Department of Defense (DoD) developed in 75 and 76 the Bell LaPadula and Biba models (respectively for confidentiality and integrity).

Let us explain the Biba model as an example (the Bell LaPadula model can be seen as a dual model of Biba). On the one hand, Biba associates integrity levels to objects and subjects. Basically, the integrity level *is(s)* of a subject *s* reflects the trust that we have in *s*; and the integrity level *io(o)* reflects the criticality of *o* and corresponds to the level of integrity of the subject that created this object. On the other hand, users and objects belong to compartments like: nuclear, medical, administrative.

The security objectives of Biba are:

- forbid any information flow from objects, having a specific level of integrity, to objects having a lower level of integrity, and
- forbid subjects modifying objects having a higher level of integrity.

As with Bell-La Padula security model, the Biba model defines a Simple Security Property and a * (star) property.

Simple security property (no-read down):

- A subject can observe an object only if the integrity label of the object is higher than the integrity label of the subject: (si, oj, observe) ➔ is(si) <= io(oj).

Star security property (no-write up):

- A subject can modify an object only if the integrity label of the subject is higher than the integrity label of the object: (si, oj, modify) ➔ io(oj) <= is(si).

#### 5.2.2.3 Role Based Access Control (RBAC)

In Role based access control model (RBAC), an intermediate entity, called "role" is added between subjects and actions. In this way, the security policy does not directly grants permissions to users but to roles. A given user will obtain permissions by playing roles (see Figure 31).

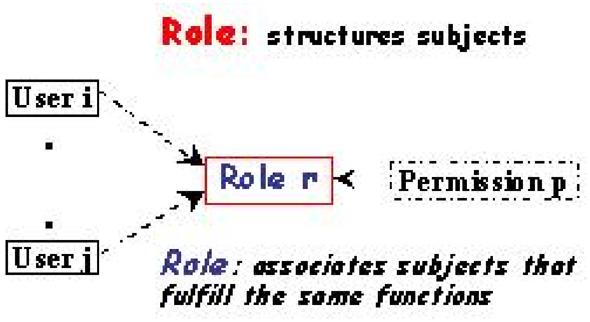
Role: structures subjects
User i
User j
Role r
Permission p
Role: associates subjects that fulfill the same functions

**Figure 31: The RBAC model.**

The initial version of RBAC was refined by including the concepts of session and role hierarchy. Within a session, a user is not obliged to activate all his/her roles but only the subset of his/her roles that are necessary to perform a given task. The role hierarchy is useful because permissions are inherited through this hierarchy. This simplifies the security policy specification. In the complete model, it is also possible to specify constraints, e.g., to take the separation of duty into account.

The RBAC model has the following drawbacks. First, the concept of permission is primitive. When specifying the security policy of a given application, the RBAC model must be refined to make explicit the structure of permissions. Second, the concept of role hierarchy is not free of ambiguity. In particular, it is generally incorrect to consider that it corresponds to an organizational hierarchy. For instance, the director of a hospital is the head of the physicians of this hospital. However, director is an administrative position and it is not required to be a physician to hold this position. Therefore, it would be incorrect to consider that the role director inherits all the permissions of the role physician. Third, it is not possible in the RBAC model to specify a permission that depends on a given context. More precisely, if a given permission is granted to a given role, then all users that play this role will inherit the given permission. Therefore, it is not possible to specify that a physician is prohibited to have a direct access to the medical records of a patient, unless the latter is one of the physician's patients. Moreover, another limit of the RBAC model is that it only enables the administrator to specify permissions. Finally, RBAC is not intended to manage the collaboration between several heterogeneous organizations.

### 5.2.3 Organization Based Access Control

Or-BAC model introduces the concept of organization. An organization can be defined as an organized group of active entities.

**Hierarchy and inheritance:**

Or-BAC makes it possible to break down an organization into several sub-organizations. Thus, we can define an organization hierarchy. If we consider a bank called *Trusted bank*, it is most likely that this organization is divided into departments, and the departments into units for example. The different agencies of this bank may also be viewed as sub-organizations. Or-BAC can model such structures. Using inheritance mechanism, if a security policy is defined for an organization, we can express that the sub-organizations may inherit of that policy. In each sub-organization, it is possible to add specific authorization rules such that the lower we are in the organization hierarchy, the more specific the security policy is. The entity organization is central in a security policy based on the Or-BAC model. All the entities presented are always defined in a given organization.

### The model entities

The Or-BAC model use the traditional entities *Subject*, *Action* and *Object* used in most access control models. The set of subjects – respectively actions and objects – are written *S* – respectively *A* and *O*. OrBAC brings in more abstraction in order to hold a large number of policies and to obtain stable policies for a long period of time. We also aim at specifying general policies, that is, policies independent from the concrete issues that might be implemented differently in several organizations.

#### Subjects and roles

**Subject:** the active entity "subject" can be either a user, i.e. a human, or an organization (or sub-organization). In fact, we assume that the set of organizations is a subset of the set of subjects, so that $Org \subseteq S$. Examples of subjects therefore include users such as *John*, *Mary*, *Peter*, etc., or organizations such as *Trusted bank* or its financial department called *Trusted finance*.

**Role:** the entity *Role is* as an abstraction of entity *Subject*. It is used to structure the link between subjects and organizations. *R* is the set of roles defined in a policy. A role consists of a set of authorizations. Assigning a subject to a role amounts to grant a set of privileges to this subject.

The role is an organizational concept. In other words, a role is defined within an organization. To model such a relation, the relationship *Relevant role* is introduced:

Relation Relevant role:

"*Relevant role* is a relation over domains $Org \times R$. If *org* is an organization and *r* a role, then *Relevant role*(*org, r*) means that *r* is a relevant role in organization *org*".

Therefore, a role can be considered as relevant in several organizations. The relation is useful in particular with respect to the hierarchies and the associated inheritance rules. Since subjects play roles in organizations, we need a relationship that joins up these entities together: the relationship *Empower*.

Relation Empower:

"*Empower* is a relation over domains $Org \times S \times R$. If *org* is an organization, *s* a subject and *r* a role, then *Empower*(*org, s, r*) means that *org* empowers subject *s* in role *r*".

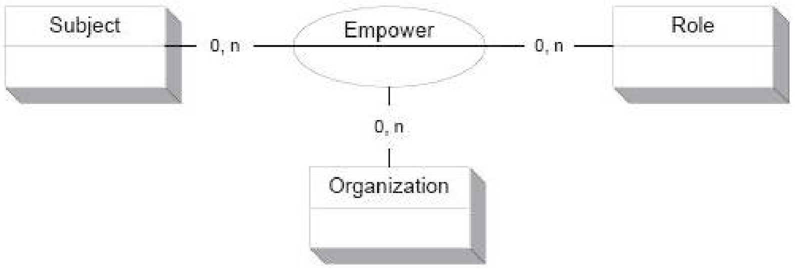

**Figure 32: The Empower relation.**

We are then able to specify that a given role carry different sets of authorizations according to the organization in which this role is defined. For instance, in the context of our bank example, a subject can play the role counter clerk in two banks and get two different sets of authorizations.

**Example**

We assume that roles *counter clerk* and *financial department* are relevant roles in organization *Trusted bank*. This is expressed this way:

• *Relevant role*(*Trusted bank, counter clerk*)

• *Relevant role*(*Trusted bank, financial department*)

The two following relations correspond to the fact that organization *Trusted bank* empowers some subjects in these roles:

• *Empower*(*Trusted bank, John, counter clerk*)

"*Trusted bank* empowers John as a counter clerk".

• *Empower*(*Trusted bank, trusted finance, financial department*)

"*Trusted bank* empowers organization *trusted finance* as its financial department".

**Objects and views**

**Object:** the entity *Object* covers data container entities such as data files, e-mails, printed forms, etc. In the banking domain, we can consider for example objects like customer accounts or company accounts. We assume that the set of subjects is a subset of the set of objects, so that $S \subseteq O$. By means of the entity *Role*, we are able to structure the subjects and to update easily the security policies when new subjects are added to the system. Since we will also have to structure the objects and to add new objects to the system, we believe that a similar entity regarding objects is necessary: the entity *View*.

**View:** a view corresponds to a set of objects that satisfy a common property. Once a view is defined in a policy, it is considered as an element of this policy, as it is for a role. A view can also be defined as the "role played" by an object, or a set of objects, within a given organization. However, we prefer to define a view from the access control standpoint only: "A *view* is an access control entity defined within an organization, and used to put together objects to which apply the same authorizations". *V* is the set of views defined in a policy. The view, like the role, is an organizational concept.

Therefore a new relation, *Relevant view*(*org, v*), that brings together a view with an organization is introduced.

<u>Relation Relevant view:</u>

"*Relevant view* is a relation over domains *Org* ×*V*. If *org* is an organization and *v* a view, then *Relevant view*(*org, v*) means that *v* is a relevant view in organization *org*".

In a file management system of a bank, for instance, the view *customer account* corresponds to the customer accounts whereas the view *company account* corresponds to the company accounts. Seeing that views characterize the ways objects are used in organizations, we need a relationship that links together these entities: the relationship *Use.*

Relation Use:

“*Use* is a relation over domains *Org* × *O* ×*V*. If *org* is an organization, *o* is an object and *v* is a view, then *Use*(*org, o, v*) means that *org* uses object *o* in view *v*”. The aim is to be able to characterize organizations that give different definitions to the same view.

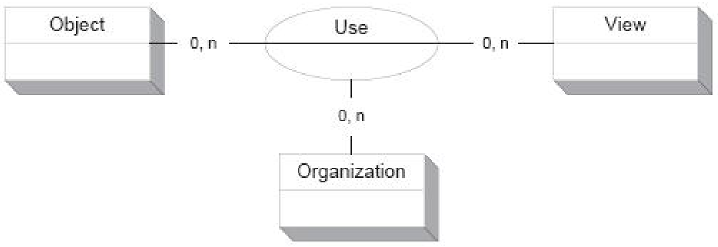

**Figure 33: The Use relation.**

**Example:**

Take the case of the view *customer account* relevant in organization *Trusted bank* and *Gold bank*, and defined as a set of Excel documents in *Trusted bank* and as a set of XML documents in *Gold bank*:

• *Use*(*Trusted bank, customer* 12*.xls, customer account*)

“*Trusted bank* uses object *customer* 12*.xls* as a customer account”.

• *Use*(*Gold bank, customer* 15*.xml, customer account*)

“*Gold bank* uses object *customer* 15*.xml* as a customer account”.

**Actions and activities**

**Action:** security policies specify the authorized accesses granted to active entities on inactive entities and regulate the actions carried out in the system. The entities *Action* will mainly contain computer actions such as *read*, *write*, *send*, etc. The set of actions *A* is a subset of the set of objects *O*, such that $A \subseteq O$.

**Activity:** subjects are abstracted by means of roles, a new entity will also be used to abstract actions: the entity *Activity*. An activity is a task, whereas an action is the way this task is executed in the organization. As for the notion of view, an activity can be seen as the “role played” by an action, or a set of actions, within an organization. Intuitively, an *activity* is an operation that can be fulfilled by some actions defined in this organization. However, we define the entity *Activity* from the security policy point of view only. “An *activity* is an access control entity defined within an organization, and used to put together some actions on which apply the same authorizations”. *A* is the set of activities defined in the policy. The activity, like the role and the view, is an organizational concept. Therefore a new relation, *Relevant activity*, that joins up an activity with an organization is introduced.

Relation Relevant activity:

“*Relevant activity* is a relation over domains *Org* × *A*. If *org* is an organization and *a* an activity, then *Relevant activity* (*org, a*) means that *a* is a relevant activity in organization *org*”.

In OrBAC, activities like *updating*, *consulting*, etc., will be of the utmost interest. Since different organizations may decide that a single action comes under distinct activities, we introduce a new relation that will be used to join up the entities *Organization*, *Action* and *Activity*: the relationship *Consider.*

Relation Consider:

"*Consider* is a relation over domains *Org* × *A* × *A*. More precisely, if *org* is an organization, *α* is an action and *a* is an activity, then *Consider*(*org, α, a*) means that *org* considers that action *α* falls within the activity *a*".

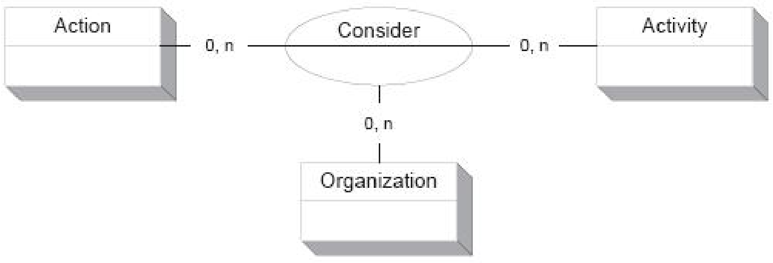

**Figure 34: The Consider relation.**

Since "*Consider"* is a ternary relation, different organizations may decide that a single action comes under distinct activities or that different actions come under the same activity. What we have in mind is to be able to characterize organizations that structure similar activities in different ways.

**Example**

We could consider, for instance, the activity *consulting* which is relevant in organizations *Trusted bank* and *Gold bank*. In the first bank, this activity might correspond to an action *read* run on data files whereas it might correspond, in the other bank, to action *select* performed on relational databases:

• *Consider*(*Trusted bank, read, consulting*)

"*Trusted bank* considers *read* as an activity consulting" and

• *Consider*(*Gold bank, select, consulting*)

"*Gold bank* considers *select* as an activity consulting".

**Attributes**

Attributes enable us to express further information about the entities involved in a security policy. This is indeed convenient to specify entity attributes in order to design expressive rules. More precisely, as shown in the following example, attributes can be used to design automatic assignment. Furthermore, attributes will be used to design contexts.

In Or-BAC every object may have some attributes. Since $S \subseteq O$ and $A \subseteq O$, attributes can actually be assigned to the concrete entities *subject* and *action*. Let us consider the following relation and assume that *account* 12 is an object: • *client type*(*account* 12*, company*)

This relation corresponds to an attribute “client type” and means that object *account* 12 is of the type *company*. Thanks to this attribute, we are able to express the following rule:

“Every object used in the view *account* having attributes *client type* equal to *company* is automatically used in the view *company account*. This is modeled with the following rule:

• $\forall s \in S$,

*Use*(*Trusted bank, s, company account*)

$\leftarrow$ *Use*(*Trusted bank, s, account*) $\wedge$ *client type*(*s, company*).

It might be useful for example to create an attribute *account balance* in order to automatically assign credit accounts in a given view and debit accounts in another view. With respect to subjects, one might want to indicate the age or the gender of the customers. Afterwards, we use, among others, the following attribute examples:

• *account number*(*number, account*) where *account* is an account and *number* is an account number.

• *account owner*(*subject, number*) where *subject* is a customer and *number* is an account number.

The definition of attributes depends of the application domain of the security policy. This is the reason why we do not specify predefined attributes.

**Organizational authorization**

Figure 35 shows the abstraction made of the entities *Subject*, *Action* and *Object* into the entities *Role*, *Activity* and *View*. We call “concrete entities” the first three entities and “organizational entities” the last three entities.

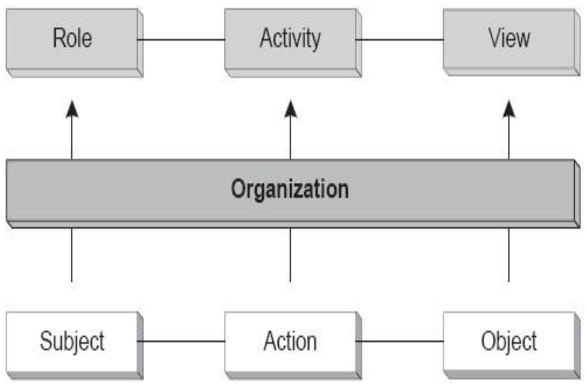

**Figure 35: Abstraction of the traditional access control entities.**

What we have in mind is to extend OrBAC with two new relations *Permission* and *Prohibition* for the purpose of being able to join together organizations, roles, views and activities and thereby expressing authorizations.

The first relation corresponds to a positive authorization, the second relation to a negative authorization.

The use of explicit negative authorizations in security policies is often discussed, mainly for two reasons. Firstly, if we go on the closed policy assumption, explicit negative authorizations might seem to be useless. Secondly, some conflicting pair of positive and negative authorizations may appear in a policy, which enables to express positive and negative authorizations at the same time.

Here is the definition of relation *Permission*. Relation *Prohibition* is defined in the same manner.

Relation Permission (first definition):

"*Permission* is a relation over domains *Org* × *R* × *A* × *V*. More precisely, if *org* is an organization, *r* is a role, *v* is a view and *a* is an activity, then *Permission*(*org, r, a, v*) means that organization *org* grants role *r* the positive authorization to perform activity *a* on view *v*".

Permissions – respectively prohibitions – handle organizational entities (roles, activities and views), they are called "organizational permissions" – respectively "organizational prohibitions".

**Example**

Let us take the case of bank *Trusted bank* that grants role *financial adviser* permission to perform activity *creation* on view *customer account*. This is expressed by the following fact:

• *Permission*(*Trusted bank, financial adviser, creating, customer account*)

Otherwise, the following fact:

• *Permission*(*Trusted bank, counter clerk, consulting, company account*) says that bank *Trusted bank* grants role *counter clerk* permission to perform activity *consulting* on view *company account*.

So far, the introduction of the entity organization and the three organizational entities allow expressing high-level security rules. Let us consider again the previous permission, which shows the benefit of such modelling. With only one security rule, *Trusted bank* can express that in all agencies and whatever IT system is used, all counter clerks are allowed to realize any action that falls within activity *consulting* on any company account.

**Modelling contexts**

An expressive policy model should provide means to define dynamic security policies, that is, policies that may depend on some specific circumstances. More precisely, it should be possible to make each security rule constrained by the policy context. In order to address this issue, we introduced a new entity in OrBAC:

***Context*:** this entity will be used to specify the concrete circumstances in which organizations grant roles some permission to perform activities on views.

Let us first consider the following permission:

• *Permission*(*Trusted bank, customer, consulting, customer account*)

"*Trusted bank* allows its customers to consult the costumer accounts".

Thanks to this privilege, the bank can express that its customers are granted the right to consult the bank accounts. On the other hand, it is clear that this security requirement is not exactly what *Trusted bank* wants to specify: a customer should be allowed to consult only his own bank account. The truth of the matter is that the Or-BAC model described above simply cannot cope with such security requirement: given an organization, users inherit permissions from the roles they play in that organization. The use of the context entity will allow the bank to express the privilege it wants.

**Definition** Context

"A *context*, within an organization, is a set of constraints that must be satisfied in order to activate an authorization".

*C* is the set of contexts defined in the policy. The context is an organizational notion. A context is actually defined within an organization. In order to model this link, we introduce a new relation *Relevant context*:

Relation Relevant context:

"*Relevant context* is a relation over domains *Org* × *C*. If *org* is an organization and *c* a context, then *Relevant context*(*org, c*) means that *c* is a relevant context in organization *org*".

Each context is seen as a ternary relation between subjects, objects and actions defined within such or such organization. Therefore, entities *Organization*, *Subject*, *Object*, *Action* and *Context* are linked together by the relationship *Hold*.

Relation Hold:

"*Hold* is a relation over domains *Org* × *S* × *A* × *O* × *C*. If *org* is an organization, *s* is a subject, *o* is an object, *α* is an action and *c* a context, then *Hold*(*org, s, α, o, c*) means that within organization *org*, context *c* is true between subject *s*, action *α* and object *o*".

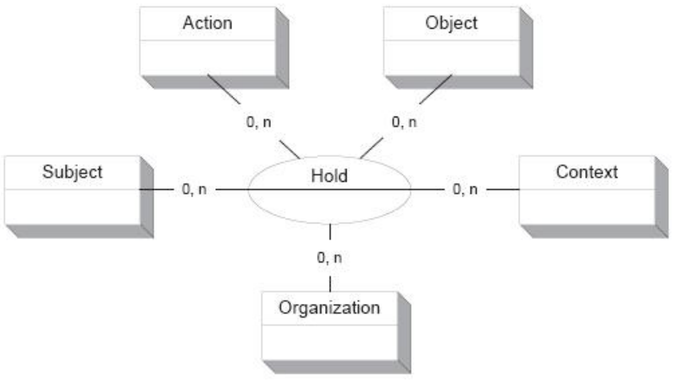

**Figure 36: The Hold relation.**

The conditions required for a given context to be linked, within a given organization, to subjects, objects and actions is formally specified by logic rules.

Relation Permission (final definition):

"*Permission* is a relation over domains *Org* × *R* × *A* × *V* × *C*. More precisely, if *org* is an organization, *r* is a role, *v* is a view, *a* is an activity and *c* is a context then *Permission*(*org, r, a, v, c*) means that organization *org* grants role *r* the positive authorization to perform activity *a* on view *v* in context *c*".

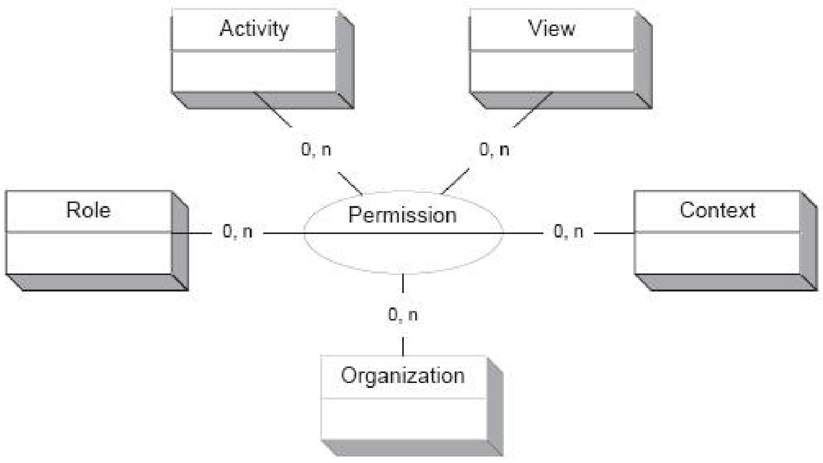

**Figure 37: The Permission relation.**

### Example

Let us get back to our previous example. *Trusted bank* can define a new context, called *personal account*:

• $\forall s \in S,\ \forall \alpha \in A,\ \forall o \in O$

*Hold*(*Trusted bank, s, α, o, personal account*) ← *account number*(*number, o*) ∧ *account owner*(*s, number*)

Using this context definition the bank is able to fully express the privilege corresponding to the permission granted to a customer to consult his personal bank account:

• *Permission*(*Trusted bank, customer, consulting, customer account, personal account*).

### Concrete permission

The relationship *Permission* enables a given organization to specify permissions granted in a given context. Such permissions correspond to a relation between roles, views and activities. However, down-to-earth access control must provide a framework to describe the concrete actions that may be performed by subjects on objects. For the purpose of modeling concrete permissions, we introduce in OrBAC the new relation *Is permitted* as a relationship between subjects, objects and actions (see Figure 38).

Is permitted:

"*Is permitted* is a relation over domains $S \times A \times O$. If $s$ is a subject, $o$ is an object and $\alpha$

is an action then *Is permitted*($s, \alpha, o$) means that subject $s$ is permitted to perform action $\alpha$ on object $o$".

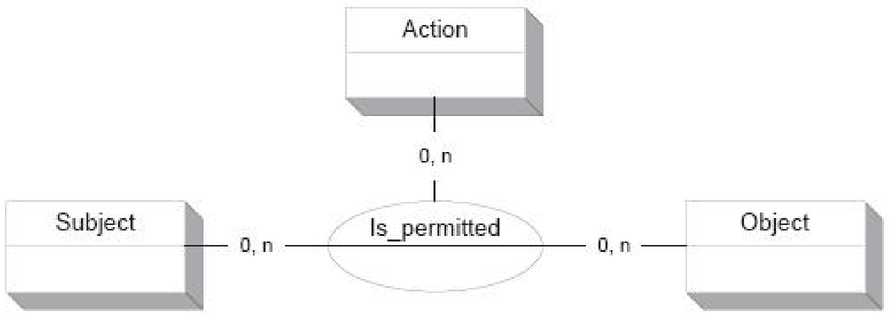

**Figure 38: The Is_permitted relation.**

We define in the same way the relationship *Is prohibited* which corresponds to a negative authorization.

In OrBAC, triplets that are instances of the relationship *Is permitted* are logically derived from organizational permissions granted to roles, views and activities by the relationship *Permission*. This is modelled by a general rule RG1.

**Example** Let us consider the following concrete permission example:

• *Is permitted*(*John, ATM.consult, account n◦*428)

“John is allowed to consult the bank account n◦428 from an ATM machine.”

Explicit instances of the relationship *Is permitted* may also be viewed as exceptions to the general security policy specified by the relationship Permission. Let us consider the following example. Usually subjects who are empowered as counter clerk are not allowed to consult the company accounts. Therefore, no organizational authorization is stated. However, and whatever could be the reason, if the bank wants to grant the permission to a specific user, for example *Paul*, to consult a specific object company account, for example *society*12*.act*,

It can add the corresponding concrete authorization: • *Is permitted* (*Paul, read, society*12*.act*)

Thereby, this concrete authorization can be viewed as an exception to the organizational policy. This is called an “explicit concrete authorization”. Actually, this notion is close to the one of delegation.

**Security policy**

An Or-BAC policy can be seen as a two level security policy: on the one hand an organizational level compound of organizational entities – *Organization*, *Role*, *Activity*, *View*, *Context* – and of organizational authorizations – *Permission*, *Prohibition* – and on the other hand a concrete level compound of concrete entities – *Subject*, *Action*, *Object* – and of concrete authorizations – *Is permitted*, *Is prohibited*.

These two levels are related as follows. In a given organization *org*, a subject *s* is permitted to perform an action *α* on an object *o* if:

i. *s* is empowered to play a given role *r* in *org* and

ii. *α* implements a given activity *a* in *org* and

iii. *o* is used in a given view *v* by *org* and

iv. a context *c* holds in *org* between *s*, *α* and *o*, and

v. the organization *org* grants to role *r* the permission to perform the activity *a* on the view *v* in the context c.

If these conditions are fulfilled, the request made by the subject *s* to perform the action *α* on the object *o* is accepted. This is modeled by the following derivation rule which makes it possible to derive concrete positive authorizations from organizational positive authorizations:

• RG1: $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall r \in R, \forall a \in A, \forall v \in V, \forall c \in C,$

*Permission*(*org, r, v, a, c*) $\wedge$

*Empower*(*org, s, r*) $\wedge$

*Consider*(*org, α, a*) $\wedge$

*Use*(*org, o, v*) $\wedge$

*Hold*(*org, s, α, o, c*)

$\rightarrow$ *Is permitted*(*s, α, o*)

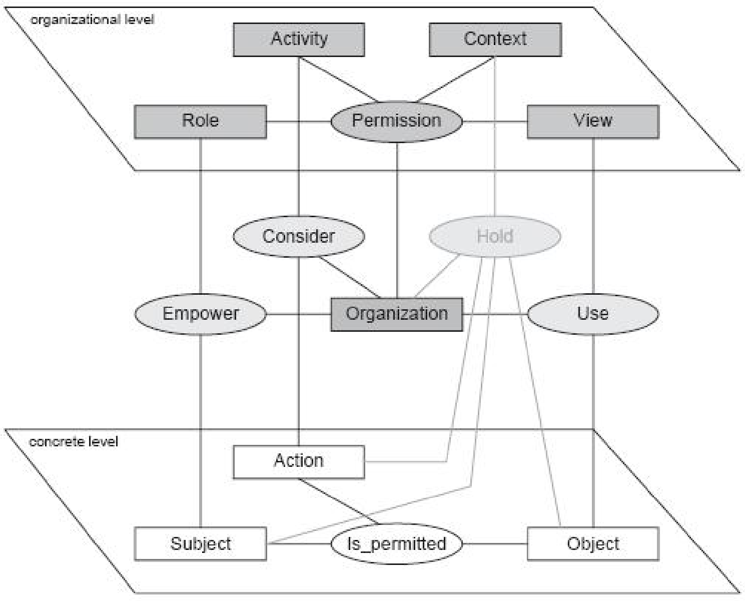

**Figure 39: The OrBAC model.**

The rule RG2 that enables to get the concrete negative authorizations from the organizational negative authorizations is stated in the same manner as rule RG1. To give a comprehensive description of Or-BAC, relations *Prohibition* and *Is-prohibited*, as well as the relevance relations, are absent of Figure 39.

Nevertheless, we suggest that concrete permissions are security rules dynamically derived. On the one hand, organizational permissions compose the written security policy. In this perspective, the derivation rules previously stated are only used when security requests are launched.

**Example**

Let us consider the following example where John, a *Trusted bank* customer, wants to consult his cash account balance on an ATM machine. The corresponding security request made after John has logged on the machine could be expressed as follows:

• *Is permitted*(*John, ATM.consult, account n*◦428)

John is a bank customer:

• *Empower*(*Trusted bank, John, customer*)

We assume that John's account number is 428:

• *Use*(*Trusted bank, account n*◦428*, customer account*)

• *Hold*(*Trusted bank, John, ATM.consult, account n*◦428*, personal account*)

We also assume that in the IT system, *ATM.consult* is the name given to the action of consulting using an ATM machine: • *Consider*(*Trusted bank, ATM.consult, consulting*).

Finally, we assume that the following organizational permission is part of the *Trusted bank* security policy:

• *Permission*(*Trusted bank, customer, consulting, customer account, personal account*)

Using these facts and the derivation rule RG1, we are able to conclude that John is allowed to consult his account balance on an ATM machine.

### 5.2.4 XACML

XACML [OASIS 2005b] is an initiative to develop a standard for access control and authorization systems. A typical access control and authorization scenario includes three main entities -- subject, resource, action -- and their attributes. A subject makes a request for permission to perform an action on a resource. For example, in the access request, "Allow the finance manager to create files in the invoice folder on the finance server," the subject is the "finance manager," the target resource is the "invoice folder on the finance server," and the action is "create files."

In proprietary access control systems, information about these entities and their attributes is kept in repositories. The repositories are called Access Control Lists (ACLs). Different proprietary systems have different mechanisms for implementing ACLs, making it difficult to exchange and share information between them.

XACML aims to achieve the following objectives:

– Create a portable and standard way of describing access control entities and their attributes.
– Provide a mechanism that offers much finer granular access control than simply denying or granting access -- that is, a mechanism that can enforce some before and after actions along with "permit" or "deny" permission.

XACML is composed of many components described in Figure 40.

A request for authorization is transmitted to the Policy Enforcement Point (PEP). The PEP creates an XACML request and sends it to the Policy Decision Point (PDP), which evaluates the request and sends back a response. The response can be either access permitted or denied, with the appropriate obligations.

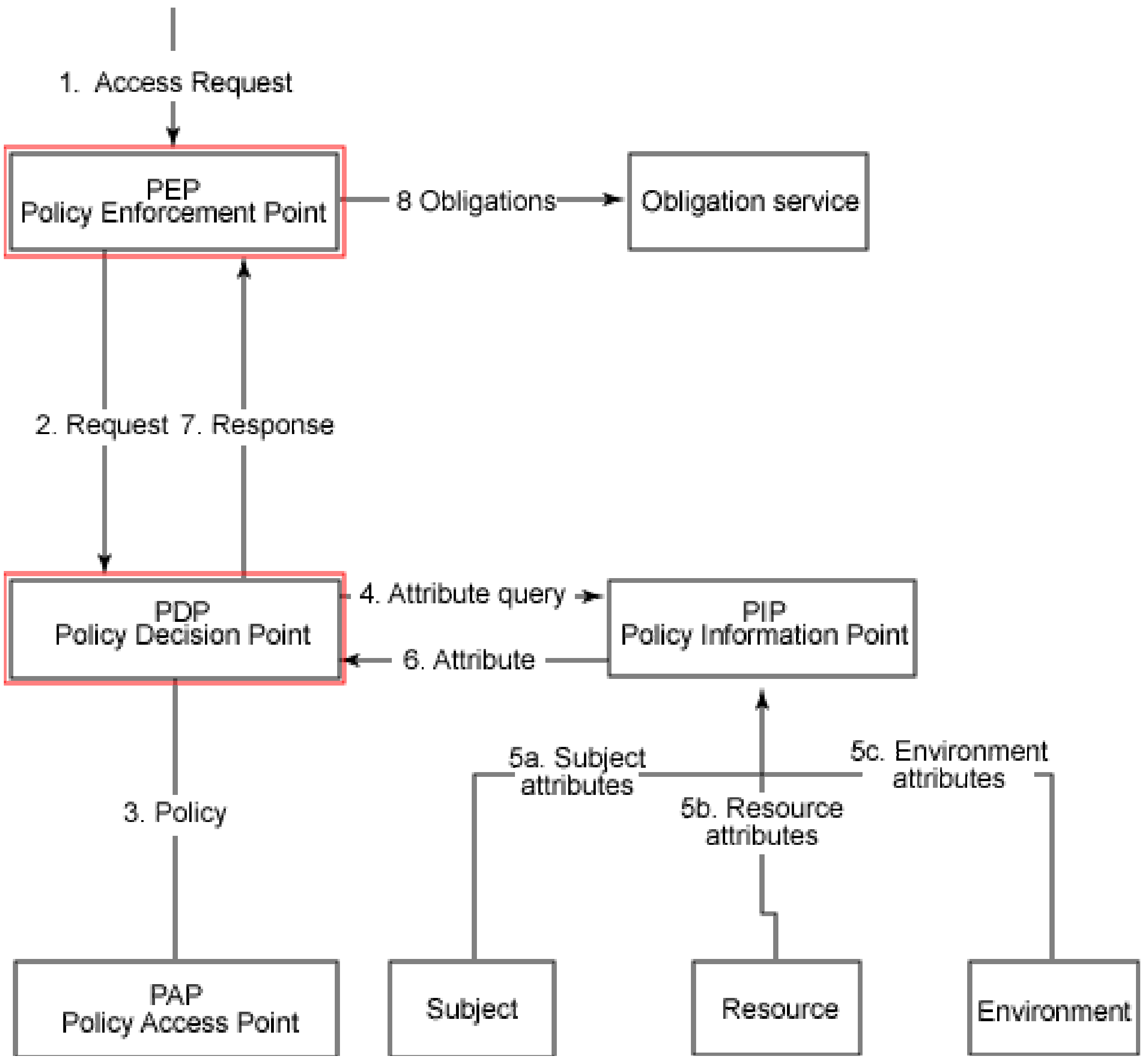

**Figure 40: The XACML main components.**

The PDP arrives at a decision after evaluating the relevant policies and the rules within them. A number of policies may be available: The PDP does not evaluate all of them; only the relevant ones are chosen for evaluation, based on the policy target. The policy target contains information about the subject, the action, and other environmental properties.

To get to the policies, the PDP uses the Policy Access Point (PAP**)**, which writes policies and policy sets, and makes them available to the PDP. The PDP may also invoke the Policy Information Point (PIP) service to retrieve the attribute values related to the subject, the resource, or the environment. The authorization decision taken by the PDP is sent to the PEP. The PEP fulfills the obligations and, based on the authorization decision sent by PDP, either permits or denies access.

## REFERENCES

[Abou El Kalam et al. 2003] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel and G. Trouessin, "Organization Based Access Control", Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks, pp.120–134, June 2003.

[Andersen et al. 2001] D. G. Andersen, H. Balakrishnan, M. Frans Kaashoek and R. Morris, "Resilient Overlay Networks", Proceedings of the 18th ACM Symposium on Operating Systems Principles, October 2001.

[Anceaume et al. 1995] E. Anceaume, B. Charron-Bost, P. Minet, and S. Toueg, "On the formal specification of group membership services", Technical Report RR-2695, INRIA, Rocquencourt, France, November 1995.

[Avizienis & Chen 1997] A. Avizienis and L. Chen, "On the implementation of N-version programming for software fault tolerance during execution", Proceedings of the IEEE COMPSAC, pp. 149–155, November 1977.

[Baldoni et al. 2003] R. Baldoni, J. Helary, M. Raynal, and L. Tanguy, "Consensus in Byzantine asynchronous systems", Journal of Discrete Algorithms, 1(2), pp. 185–210, 2003.

[Barrantes et al. 2003] E. G. Barrantes, D. H. Ackley, T. S. Palmer, D. Stefanovic, and D. D. Zovi, "Randomized instruction set emulation to disrupt binary code injection attacks", Proceedings of the 10th ACM Conference on Computer and Communications Security, pp. 281–289, 2003.

[Bellovin 1999] S. Bellovin. "Distributed Firewalls", :login;, November 1999.

[Bell & LaPadula 1976] D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation", Technical Report ESD-TR-75-306, MTR-2997, Rev. 1, MITRE Corporation, March 1976.

[Ben-Or 1983] M. Ben-Or, "Another advantage of free choice: Completely asynchronous agreement protocols", Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing, pp. 27–30, August 1983.

[Bertino et al. 2000] E. Bertino, S. Castano, E. Ferrari, and M. Mesiti, "Specifying and Enforcing Access Control Policies for XML Document Sources", World Wide Web Journal, 2000.

[Bhatkar et al. 2003] S. Bhatkar, D. C. DuVarney, and R. Sekar, "Address obfuscation: An efficient approach to combat a broad range of memory error exploits", Proceedings of the 12th USENIX Security Symposium, pp.105–120, August 2003.

[Bhatkar et al. 2005] S. Bhatkar, R. Sekar, and D. C. DuVarney, "Efficient techniques for comprehensive protection from memory error exploits", Proceedings of the 14th USENIX Security Symposium, pp.271–286, August 2005.

[Birman 1997] K. P. Birman, "Building Secure and Reliable Network Applications", Manning Publishing Company and Prentice Hall, 1997.

[Birman & Joseph 1987a] K. Birman and T. Joseph, "Exploiting virtual synchrony in distributed systems", Proceedings of the 11th Symposium on Operating System Principles, pp. 123–138, November 1987.

[Birman & Joseph 1987b] K. Birman and T. Joseph, "Reliable communication in the presence of failures", ACM Transactions on Computer Systems, 5(1), pp. 46–76, 1987.

[Bishop & Dilger 1996] M. Bishop and M. Dilger, "Checking for race conditions in file access", Computing Systems, 9(2), pp. 131–152, 1996.

[Bondavalli et al. 2000] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and F. Grandoni, "Threshold-based mechanisms to discriminate transient from intermittent faults", IEEE Transactions on Computers, 49(3), pp. 230–245, 2000.

[Bondavalli et al. 2004] A. Bondavalli, S. Chiaradonna, D. Cotroneo, and L. Romano, "Effective fault treatment for improving the dependability of COTS- and legacy-based applications", IEEE Transactions on Dependable and Secure Computing, 1(4) pp. 223-237, 2004.

[Bracha 1984] G. Bracha, "An asynchronous $\lfloor(n-1)/3\rfloor$-resilient consensus protocol", Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing, pp. 154–162, August 1984.

[Byres & Lowe 2004] E. Byres and J. Lowe, "The myths and facts behind cyber security risks for industrial control systems", Proceedings of VDE Kongress, 2004.

[Byres et al. 2005] E. Byres, J. Karsch, and J. Carter, "NISCC good practice guide on firewall deployment for SCADA and process control networks", Technical report, NISCC, February 2005.

[Cachin et al. 2000] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography", Proceedings of the 19th ACM Symposium on Principles of Distributed Computing, pp. 123–132, July 2000.

[Cachin et al. 2001] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols (extended abstract)", In J. Kilian, editor, Advances in Cryptology: CRYPTO 2001, volume 2139 of Lecture Notes in Computer Science, pp. 524–541, 2001.

[Cachin et al. 2002] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl, "Asynchronous verifiable secret sharing and proactive cryptosystems", Proceedings of the 9th ACM Conference on Computer and Communications Security, pp.88–97, 2002.

[Casimiro et al. 2000] A. Casimiro, P. Martins, and P. Veríssimo, "How to build a Timely Computing Base using Real-Time Linux", Proceedings of the IEEE International Workshop on Factory Communication Systems, pp. 127–134, September 2000.

[Casimiro & Verissimo 2001] A. Casimiro and P. Veríssimo, "Using the Timely Computing Base for Dependable QoS Adaptation", Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems , October 2001

[Castro & Liskov 2002] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery", ACM Transactions on Computer Systems, 20(4), pp. 398–461, November 2002.

[Chandra et al. 1996] T. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost, "On the impossibility of group membership", Proceedings of the 15th ACM Symposium on Principles of Distributed Computing, pp. 322–330, May 1996.

[Chandra & Toueg 1996] T. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems", *Journal of the ACM*, 43(2), pp. 225–267, March 1996.

[Chen et al. 2006] S. Chen, J. Xu, Z. Kalbarczyk, and K. Iyer, “Security vulnerabilities: From analysis to detection and masking techniques”, Proceedings of the IEEE, 94(2), pp. 407–418, February 2006.

[Christian & Fetzer 1998] F. Christian and C. Fetzer, “The timed asynchronous system model”, Proceedings of the 28th IEEE International Symposium on Fault-Tolerant Computing, pp. 140–149, 1998.

[Cieslewicz 2004] J. Cieslewicz, “Attacks and accidents: Policy to protect the power grid’s critical computing and communication needs”, Senior interdisciplinary honors thesis in international security studies, Stanford University, May 2004.

[Collins 1998] R. R. Collins, “The Pentium F00F bug”, Dr. Dobb’s Journal of Software Tools, 23(5), pp. 62, 64–66, May 1998.

[Correia et al. 2002] M. Correia, P. Veríssimo, and N. F. Neves, “The design of a COTS real-time distributed security kernel”, Proceedings of the Fourth European Dependable Computing Conference, pp. 234–252, October 2002.

[Correia et al. 2002b] M. Correia, L. C. Lung, N. F. Neves, and P. Veríssimo, “Efficient Byzantine-resilient reliable multicast on a hybrid failure model”, Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems, pages 2–11, October 2002.

[Correia et al. 2004] M. Correia, N. F. Neves, and P. Veríssimo, “How to tolerate half less one Byzantine nodes in practical distributed systems“, Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems, October 2004.

[Correia et al. 2005] M. Correia, N. F. Neves, L. C. Lung, and P. Veríssimo, “Low complexity Byzantine-resilient consensus”, Distributed Computing, 17(3), pp. 237–249, 2005.

[Correia et al. 2006a] M. Correia, N. F. Neves, and P. Veríssimo, “From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures”, Computer Journal, 41(1), pp. 82–96, January 2006.

[Correia et al. 2006b] M. Correia, N. F. Neves, L. C. Lung and P. Veríssimo, “Worm-IT – A Wormhole-based Intrusion-Tolerant Group Communication System“, Journal of Systems and Software. 2006. *To appear.*

[Cowan et al. 2000] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole, “Buffer overflows: Attacks and defenses for the vulnerability of the decade”, DARPA Information Survivability Conference and Expo, January 2000.

[Crutial_D2 2007] CRUTIAL Project, “Analysis of new control applications”, CRUTIAL project, Deliverable D2, January 2007.

[Crutial_D3 2007] CRUTIAL Project, “Methodologies synthesis”, CRUTIAL project, Deliverable D3, January 2007.

[Cuppens 2004] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège, “A formal approach to specify and deploy a network security policy”, Second Workshop on Formal Aspects in Security and Trust, August 2004.

[Daidone et al. 2006] A. Daidone, F. Di Giandomenico, A. Bondavalli, “Hidden Markov Models as a support for diagnosis: formalization of the problem and synthesis of the solution”, Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems, October 2006.

[De Florio & Deconinck 2002] V. De Florio, G. Deconinck, “REL: A Fault-Tolerance Linguistic Structure for Distributed Applications”, Proceedings 9th IEEE Conference and Workshop on Engineering of Computer-Based Systems, pp. 51-58, April 2002.

[Deconinck et al. 2002] G. Deconinck, V. De Florio, O. Botti, “Software-Implemented Fault Tolerance and Separate Recovery Strategies Enhance Maintainability”, IEEE Transactions on Reliability, 51(2), pp. 158-165, June 2002.

[Deconinck et al. 2003] G. Deconinck, V. De Florio, R. Belmans, G. Dondossola, J. Szanto, “Experiences with integrating recovery strategies into a primary substation automation system”, Proceedings of the International Conference on Dependable Systems and Networks, pp. 80-85, June 2003.

[Deswarte et al. 1991] Y. Deswarte, L. Blain, and J. C. Fabre, “Intrusion tolerance in distributed computing systems”, Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy, pp.110–121, May 1991.

[Deswarte et al. 1998] Y. Deswarte, K. Kanoun, and J.-C. Laprie, “Diversity against accidental and deliberate faults”, In P. Ammann, B. H. Barnes, S. Jajodia, and E. H. Sibley, editors, Computer Security, Dependability, and Assurance: From Needs to Solutions, pp. 171–181, November 1998.

[Dierks & Allen 1999] T. Dierks and C. Allen, “The TLS Protocol, Version 1.0,”, Network Working Group, RFC 2246, January 1999.

[Dolev et al. 1987] D. Dolev, C. Dwork, and L. Stockmeyer, “On the minimal synchronism needed for distributed consensus”, Journal of the ACM, 34(1), pp. 77–97, January 1987.

[Dondossola et al. 2006] G. Dondossola, G. Deconinck, F. Di Giandomenico, S. Donatelli, M. Kaaniche, and P. Veríssimo, “Critical utiliy infrastructural resilience”, *International Workshop on Complex Network and Infrastructure Protection*, March 2006.

[Doudou & Schiper 1997] A. Doudou and A. Schiper, “Muteness detectors for consensus with Byzantine processes”, Technical Report 97/30, EPFL, 1997.

[Doudou et al. 2002] A. Doudou, B. Garbinato, and R. Guerraoui, “Encapsulating failure detection: From crash-stop to Byzantine failures”, International Conference on Reliable Software Technologies, pp. 24–50, May 2002.

[Dwork et al. 1988] C. Dwork, N. Lynch, and L. Stockmeyer, “Consensus in the presence of partial synchrony”, Journal of the ACM, 35(2), pp. 288–323, April 1988.

[Dzung et al. 2005] D. Dzung, M. Naedele, T. P. V. Hoff, and M. Crevatin, “Security for industrial communication systems”, Proceedings of the IEEE, 93(6), pp. 1152–1177, 2005.

[Ferraiolo & Kuhn 1992] D. F. Ferraiolo and R. Kuhn, “Role-Based Access Controls”, In Z. Ruthberg and W. Polk, editors, Proceedings of the 15th NIST-NSA National Computer Security Conference, pp. 554–563, October 1992.

[Fetzer 2003] C. Fetzer, “Perfect failure detection in timed asynchronous systems”, IEEE Transactions Computing, 52(2), pp. 99–112, 2003.

[Fink and Carlsen 1978] L. H. Fink, and K. Carlsen, “Operating under stress and strain”, IEEE Spectrum, March 1978.

[Fischer et al. 1985] M. J. Fischer, N. A. Lynch, and M. S. Paterson, “Impossibility of distributed consensus with one faulty process”, *Journal of the ACM*, 32(2), pp. 374–382, April 1985.

[Forrest et al. 1997] S. Forrest, A. Somayaji, and D. H. Ackley, “Building diverse computer systems”, Proceedings of the 6th Workshop on Hot Topics in Operating Systems, pp. 67–72, May 1997.

[Friedman et al. 2005] R. Friedman, A. Mostefaoui, and M. Raynal, “Building and using quorums despite any number of process crashes”, Proceedings of the 5th European Dependable Computing Conference, April 2005.

[Friedman et al. 2005b] R. Friedman, A. Mostefaoui, and M. Raynal, “Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems”, IEEE Transactions on Dependable and Secure Computing, 2(1), pp. 46–56, January 2005.

[Frier et al. 1996] A. Frier, P. Karlton, and P. Kocher, “The SSL 3.0 protocol”, Netscape Communications Corp., November 1996.

[Gabillon 2004] A. Gabillon, “An Authorization Model for XML DataBases”, ACM Workshop on Secure Web Services, October, 2004.

[Garay et al. 2000] J. A. Garay, R. Gennaro, C. Jutla, and T. Rabin, “Secure distributed storage and retrieval”, Theoretical Computer Science, 243(1-2), pp. 363–389, 2000.

[Geer 2006] D. Geer, “Security of critical control systems sparks concern”, *IEEE Computer*, pp. 20-23, January 2006.

[Gong 1992] L. Gong, “A Security Risk of Depending on Synchronized Clocks”, Operating Systems Review, 26(1), pp. 49-53, 1992.

[Gordon et al. 2006] L. A. Gordon, M. P. Loeb, W. Lucyshyn, and R. Richardson, “CSI/FBI computer crime and security survey”, Computer Security Institute, 2006.

[Guiri 1995] L. Guiri, “A new model for role-based access control”, Proceedings of the 11th AnnualComputer Security Applications Conference, pp. 249–255, December 1995.

[Hadzilacos & Toueg 1994] V. Hadzilacos and S. Toueg, “A modular approach to fault-tolerant broadcasts and related problems”, Technical Report TR94-1425, Cornell University, Department of Computer Science, May 1994.

[Harrison et al. 1976] M. Harrison, W. Ruzzo, and J. Ullman, “Protection in operating systems”, *Communication of ACM*, 19(8), pp. 461–471, August 1976.

[Helary et al. 2000] J. M. Helary, M. Hurfin, A. Mostefaoui, M. Raynal, and F. Tronel, “Computing global functions on asynchronous distributed systems with perfect failure detectors”, IEEE Transactions on Parallel and Distributed Systems, 11(9), September 2000.

[Herzberg et al. 1997] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung, “Proactive public key and signature systems”, Proceedings of the 4th ACM Conference on Computer and Communications Security, pp.100–110,1997.

[Herzberg et al. 1995] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, “Proactive secret sharing or: How to cope with perpetual leakage”, Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology, pp.339–352, 1995.

[Hoglund & McGraw 2004] G. Hoglund and G. McGraw, “Exploiting Software: How to Break Code”, Addison-Wesley, 2004.

[ISO 1999] ISO/IEC Standard 15408, “*Evaluation Criteria for IT Security, parts 1 to 3*”, 1999.

[ISO 2005] ISO/IEC, “Common Criteria for Information Technology Security Evaluation, v3, Part 1: Introduction and general model”, ISO/IEC 15408-1, July 2005.

[Jelasity et al. 2005] M. Jelasity, A. Montresor, O. Babaoglu, “Gossip-based aggregation in large dynamic networks”, ACM Transactions on Computer Systems, 23(3), pp. 219–252, 2005.

[Joseph & Avizienis 1988] M. K. Joseph and A. Avizienis, “A fault-tolerant approach to computer viruses”, Proceedings of the 1988 IEEE Symposium on Research in Security and Privacy, pp.52–58, April 1988.

[Kalam et al. 2001] Kalam, R. Elbaida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Mige, C. Saurel, and G. Trouessin, “Organization-based access control”, *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, pp. 277-288, June 2003.

[Keidar & Rajsbaum 2001] I. Keidar and S. Rajsbaum, “On the cost of fault-tolerant consensus when there are no faults - a tutorial”, SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory), 32(2), pp. 45–63, 2001.

[Kent & Atkinson 1998a] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998. (superseded by RFC 4301)

[Kent & Atkinson 1998b] Kent, S. and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998. (superseded by RFCs 4302 and 4305)

[Kent & Atkinson 1998c] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998. (superseded by RFCs 4303 and 4305)

[Kihlstrom et al. 2001] K. Kihlstrom, L. E. Moser and P. M. Melliar-Smith, “The SecureRing Group Communication System”, ACM Transactions on Information and System Security, 4(4), pp. 371–406, 2001.

[Kihlstrom et al. 2003] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, “Byzantine fault detectors for solving consensus”, The Computer Journal, 46(1), pp. 16–35, January 2003.

[Knight & Leveson 1986] J. C. Knight and N. G. Leveson, “An experimental evaluation of the assumption of independence in multi-version programming”, IEEE Transactions on Software Engineering, 12(1), pp. 96–109, January 1986.

[Lamport 1998] L. Lamport, “The part-time parliament”, ACM Transactions Computer Systems, 16(2), pp. 133–169, May 1998.

[Lamport 2001] L. Lamport, “Paxos made simple”, SIGACT News, 32(4), pp. 51–58, 2001.

[Lamspon 1971] B. Lampson, “Protection”, Proceedings of the 5th Princeton Symposium on Information Sciences and Systems, pp. 437–443, March 1971.

[Li et al. 2005] H. Li, G. W. Rosenwald, J. Jung, and C. Liu, “Strategic power infrastructure defense”. *Proceedings of the IEEE*, 93(5), pp. 918-933, May 2005.

[Littlewood & Strigini 2004] B. Littlewood and L. Strigini, "Redundancy and diversity in security", In P. Samarati, P. Rian, D. Gollmann, and R. Molva, editors, Computer Security – ESORICS 2004, 9th European Symposium on Research Computer Security, LNCS 3193, pp. 423–438, 2004.

[Luiijf & Klaver 2004] H. Luiijf and M. Klaver, "The current state of threats", *e-Security in Europe: Todays Status and The Next Step*, October 2004.

[Malkhi & Reiter 1997] D. Malkhi and M. Reiter, "Unreliable intrusion detection in distributed computations", Proceedings of the 10th Computer Security Foundations Workshop, pp. 116–124, June 1997.

[Madani & Novosel 2005] V. Madani and D. Novosel. "*Getting a grip on the grid*". *IEEE Spectrum*, 42(12):42-47, December 2005.

[Marsh & Schneider 2004] M. A. Marsh and F. B. Schneider, "CODEX: A robust and secure secret distribution system", IEEE Transactions on Dependable and Secure Computing, 1(1), pp. 34–47, January–March 2004.

[Martin & Alvisi 2005] J. P. Martin and L. Alvisi, "Fast Byzantine consensus", Proceedings of the IEEE International Conference on Dependable Systems and Networks, pp. 402–411, June 2005.

[Mazzini et al. 2002] G. Mazzini, G.P. Nizzoli, P. Bergamo, "Measurements of Redundant Source-Routing", Proceedings of the IEEE 10th International Conference on Software, Telecommunications and Computer Networks, pp. 95-99, October 2002.

[Miege 2005] A. Miege, "Definition of a formal framework for specifying security policies: The Or-BAC model and extensions", PhD Thesis, ENST, Paris, June 2005.

[Moniz et al. 2006a] H. Moniz, M. Correia, N. F. Neves, and P. Veríssimo, "Randomized intrusion-tolerant asynchronous services", Proceedings of the International Conference on Dependable Systems and Networks, June 2006.

[Moniz et al. 2006b] H. Moniz, M. Correia, N. F. Neves, and P. Veríssimo, "Experimental Comparison of Local and Shared Coin Randomized Consensus Protocols", Proceedings of the International Conference on Reliable Distributed Systems, October 2006.

[Moser & Melliar-Smith 1999] L. E. Moser and P. M. Melliar-Smith, "Byzantine-Resistant Total Ordering Algorithms“, Information and Computation, 150, pp. 75–111, 1999.

[Moser et al. 2000] L.M. Moser, P.M. Melliar-Smith, N. Narasimhan, "The SecureGroup communication system“, Proceedings of the IEEE Information Survivability Conference, pp. 507–516, Jan. 2000.

[NCSC 1987] NCSC, "A Guide to Understanding Discretionary Access Control in Trusted Systems", NCSC-TG-003, September 1987.

[Neves et al. 2005] N. F. Neves, M. Correia, and P. Veríssimo, "Solving vector consensus with a wormhole", IEEE Transactions on Parallel and Distributed Systems, 16(12), pp. 1120–1131, December 2005.

[Neves et al. 2006] N. Neves, J. Antunes, M. Correia, P. Veríssimo, R. Neves, "Using Attack Injection to Discover New Vulnerabilities", Proceedings of the International Conference on Dependable Systems and Networks, pp. 457-466, June 2006.

[OASIS 2005a] OASIS, "UDDI Specifications TC, Universal Description, Discovery and Integration", v3.0.2, February 2005.

[OASIS 2005b] OASIS, "eXtensible Access Control Markup Language (XACML) TC", v2.0, February 2005.

[Obelheiro & Fraga 2006] R. Obelheiro and J. S. Fraga, "A Lightweight Intrusion Tolerant Overlay Network", Proceedings of 9th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, April 2006.

[Obelheiro et al. 2006] R. R. Obelheiro, A. N. Bessani, L. C. Lung, and M. Correia, "How practical are intrusion-tolerant systems?",Technical Report DI/FCUL TR-06-15. September 2006.

[Ostrovsky & Yung 1991] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks (extended abstract)", Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing, pp.51–59, 1991.

[Pease et al. 1980] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults", Journal of the ACM, 27(2), pp. 228–234, April 1980.

[Percival 2005] C. Percival, "Cache missing for fun and profit", May 2005. On-line at http://www.daemonology.net/papers/htt.pdf.

[Pizza et al. 1998] M. Pizza, L. Strigini, A. Bondavalli, and F. Di Giandomenico, "Optimal discrimination between transient and permanent faults", Proceedings of the Third IEEE International High-Assurance Systems Engineering Symposium, pp. 214–223, 1998.

[Pollet 2002] J. Pollet, "Developing a solid SCADA security strategy", Proceedings of the ISA/IEEE Sensors for Industry Conference, pp. 148-156, November 2002.

[Postel 1980] J. Postel, "User Datagram Protocol", RFC 768, August 1980.

[Postel 1981a] J. Postel, "Internet Protocol", RFC 791, September 1981.

[Postel 1981b] J. Postel, "Transmission Control Protocol", RFC 793, September 1981.

[Powell 1995] D. Powell, "Failure mode assumptions and assumption coverage", Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing, pp.386–395, July 1992.

[Powell & Stroud 2003] D. Powell, R. Stroud (editors), "Conceptual Model and Architecture of MAFTIA", MAFTIA deliverable D21, 2003.

[Powell et al. 1988] D. Powell, D. Seaton, G. Bonn, P. Veríssimo, and F. Waeselynk, "The Delta-4 approach to dependability in open distributed computing systems", Proceedings of the 18th IEEE International Symposium on Fault-Tolerant Computing, June 1988.

[Powell et al. 1998] D. Powell, C. Rabéjac, and A. Bondavalli, "Alpha-Count Mechanism and Inter-Channel Diagnosis", ESPRIT Project 20716, GUARDS Report no. I1SA1.TN.5009.E, 1998.

[Rabin 1983] M. O. Rabin, "Randomized Byzantine generals", Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science, pp. 403–409, November 1983.

[Ramasamy et al. 2002] H. Ramasamy, P. Pandey, J. Lyons, M. Cukier, and W.H. Sanders, "Quantifying the cost of providing intrusion tolerance in group communication systems", Proceedings of the International Conference on Dependable Systems and Networks. pp. 229–238, June 2002.

[Ramasamy & Cachin 2005] H. V. Ramasamy and C. Cachin, "Parsimonious Asychronous Byzantine-Fault-Tolerant Atomic Broadcast", Proceedings of the 9th International Conference on Principles of Distributed Systems, December 2005.

[Randell 1975] B. Randell, "System structure for software fault tolerance", IEEE Transactions on Software Engineering, SE-1, pp. 220–232, June 1975.

[Raynal 2005] M. Raynal, "Short introduction to failure detectors for asynchronous distributed systems", SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory), 36(1), pp. 53–70, 2005.

[RdS 2006] RdS, Ricerca di Sistema nel settore elettrico, http://www.ricercadisistema.it/

[Reiter 1994] M. Reiter, "Secure agreement protocols: Reliable and atomic group multicast in Rampart", Proceedings of the 2nd ACM Conference on Computer and Communications Security, pp. 68–80, November 1994.

[Reiter 1996] M. Reiter, "A secure group membership protocol", IEEE Transactions on Software Engineering, 22 (1), pp. 31–42, January 1996.

[Rodrigues & Verissimo 2000] L. Rodrigues and P. Veríssimo, "Topology-aware algorithms for large-scale communication", In S. Krakowiak and S. Shrivastava, editors, Advances in Distributed Systems, Volume 1752 of Lecture Notes in Computer Science, Chapter 6, pp. 127-156, Springer-Verlang, 2000.

[Romano et al. 2002] L. Romano, A. Bondavalli, S. Chiaradonna, D. Cotroneo, "Implementation of Threshold-based Diagnostic Mechanisms for COTS-based Applications", Proceedings of the 21th IEEE Symposium on Reliable Distributed Systems, pp. 296–303, October 2002.

[Shacham et al. 1998] A. Shacham, R. Monsour, R. Pereira, and M. Thomas, "IP Payload Compression Protocol (IPComp)", RFC 2393, December 1998.

[Shacham et al. 2004] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization", Proceedings of the 11th ACM Conference on Computer and Communications Security, pp.298–307, October 2004.

[Shandhu 1998] R. S. Sandhu, "Role-Based Access Control", Advances in Computers, 46, 1998.

[Sandhu et al. 1996] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, "Role-Based Access Control Models", IEEE Computer, 29(2), pp. 38-47, February 1996.

[Siewiorek & Swarz 1992] D. P. Siewiorek and R. S. Swarz, "Reliable Computer Systems: Design and Evaluation (2nd Edition)", Digital Press, 1992.

[Sousa et al. 2005] P. Sousa, N. F. Neves, and P. Verissimo, "How resilient are distributed *f* fault/intrusion-tolerant systems?", Proceedings of the International Conference on Dependable Systems and Networks, pp.98–107, June 2005.

[Sousa et al. 2005b] P. Sousa, N. F. Neves, and P. Veríssimo, "Resilient state machine replication", Proceedings of the 11th Pacific Rim International Symposium on Dependable Computing, pp. 305-309, December 2005.

[Sousa et al. 2006] P. Sousa, N. F. Neves, A. Lopes, and P. Verissimo, "On the resilience of intrusion-tolerant distributed systems", DI/FCUL TR 06–14, Dep. of Informatics, Univ. of Lisbon, September 2006.

[Stamp et al. 2003] J. Stamp, J. Dillinger, W. Young, and J. DePoy, “Common vulnerabilities in critical infrastructure control systems”, Technical report, Sandia National Laboratories, May 2003.

[Stevens 1990] W. R. Stevens, “Unix Network Programming”, Prentice Hall, 1990.

[Stouffer et al. 2006] K. Stouffer, J. Falco, and K. Kent, “Guide to supervisory control and data acquisition (SCADA) and industrial control systems security”, Recommendations of the National Institute of Standards and Technology. Special Publication 800-82, NIST, September 2006.

[Thomas & Sandhu 1994] R. K. Thomas, R. S. Sandhu, “Conceptual Foundations for a Model of Task-based Authorizations”, Proceedings of the 7th IEEE Computer Security Foundations Workshop, pp. 66-79, June 1994.

[Thomas & Sandhu 1997] R. K. Thomas, R. S. Sandhu, “Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management", Proceedings of the IFIP WG11.3 Workshop on Database Security, pp. 166-181, August 1997.

[Toueg 1984] S. Toueg, “Randomized Byzantine agreements”, Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing, pp. 163–178, August 1984.

[Turner et al. 2005] D. Turner, S. Entwisle, O. Friedrichs, D. Ahmad, J. Blackbird, M. Fossi, D. Hanson, S. Gordon, D. Cole, D. Cowlings, D. Morss, B. Bradley, P. Szor, E. Chien, J. Ward, J. Gough, and J. Talbot, “Symantec Internet security threat report. Trends for January 05 - June 05”, Symantec, Volume VIII, September 2005.

[UCTE 2006] UCTE, “UCTE Operation Handbook”, https://www.ucte.org.

[US_Energy 2002] President’s Critical Infrastructure Protection Board and Office of Energy Assurance U.S. Department of Energy, “21 Steps to Improve Cyber Security of SCADA Networks”, U.S. Department of Energy, 2002.

[US-Canada 2003] US-Canada Power System Outage Task Force, “Interim Report: Causes of the August 14th Blackout in the United States and Canada”. November 2003.

[van Eeten et al. 2006] M. van Eeten, E. Roe, P. Schulman, and M. de Bruijne. “The enemy within: System complexity and organizational surprises”. In M. Dunn and V. Mauer, editors, International CIIP Handbook 2006, volume II, pp. 89-110, Center for Security Studies, ETH Zurich, 2006.

[Vanthournout et al. 2004] K. Vanthournout, G. Deconinck, R. Belmans, “A Middleware Control Layer for Distributed Generation Systems”, Proceedings of the IEEE Power Systems Conference and Exhibition, October 2004.

[Vanthournout et al. 2005a] K. Vanthournout, K. De Brabandere, E. Haesen, J. Van den Keybus, G. Deconinck, R. Belmans, “Agora: Distributed Tertiary Control of Distributed Resources”, Proceedings of the 5th Power Systems Computation Conference, August 2005.

[Vanthournout et al. 2005b] K. Vanthournout, G. Deconinck, R. Belmans, “A Taxonomy for Resource Discovery”, Personal and Ubiquitous Computing Journal, 9(2), pp. 81-89, March 2005.

[Verissimo 2003] P. Veríssimo, “Uncertainty and predictability: Can they be reconciled?”, Future Directions in Distributed Computing, volume 2584 of *Lecture Notes in Computer Science*, pp. 108–113. 2003.

[Verissimo 2006] P. Verissimo, “Travelling through wormholes: a new look at distributed systems models”, SIGACT News, 37(1), pp. 66–81, 2006.

[Verissimo & Almeida 1995] P. Veríssimo, and C. Almeida, “Quasi-synchronism: a step away from the traditional fault-tolerant real-time system models”, Bulletin of the Technical Committee on Operating Systems and Application Environments, 7(4), pp.35-39, Winter 1995.

[Verissimo & Casimiro 2002] P. Veríssimo and A. Casimiro, “The Timely Computing Base model and architecture”, IEEE Transactions on Computers, 51(8), pp. 916–930, August 2002.

[Verissimo & Rodrigues 2001] P. Veríssimo and L. Rodrigues, “Distributed Systems for System Architects”, Kluwer Academic Publishers, 2001.

[Verissimo et al. 1997] P. Veríssimo, L. Rodrigues, and A. Casimiro, “Cesiumspray: a precise and accurate global clock service for large-scale systems”, Journal of Real-Time Systems, 12(3), pp. 243–294, May 1997.

[Verissimo et al. 2000] P. Veríssimo, A. Casimiro, and C. Fetzer, “The Timely Computing Base: Timely actions in the presence of uncertain timeliness”, Proceedings of the International Conference on Dependable Systems and Networks, pp. 533–542, June 2000.

[Verissimo et al. 2003] P. Veríssimo, N. F. Neves, and M. Correia, “Intrusion-tolerant architectures: Concepts and design”, In R. Lemos, C. Gacek, and A. Romanovsky, editors, Architecting Dependable Systems, volume 2677, pp. 3-36. 2003.

[Verissimo et al. 2006] P. Veríssimo, N. F. Neves, C. Cachin, J. Poritz, D. Powell, Y. Deswarte, R. Stroud, and I.Welch, “Intrusion-tolerant middleware: The road to automatic security”, IEEE Security & Privacy, 4(4), pp. 54-62, July/August 2006.

[W3C 2004] W3C, “Extensible Markup Language (XML)”, W3C Recommendation , February 2004.

[W3C 2003] W3C, “SOAP, Version 1.2” W3C Recommendation, June 2003

[W3C 2006] W3C, “Web Services Description Language (WSDL), Version 2.0”, W3C Candidate Recommendation, March 2006.

[Walter et al. 1997] C. Walter, P. Lincoln, and N. Suri, “Formally Verified On-Line Diagnosis”, IEEE Transactions on Software Engineering, 23(11), November 1997.

[Wilson 2006] C. Wilson, “Terrorist capabilities for cyber-attack”, In M. Dunn and V. Mauer, editors, International CIIP Handbook 2006, volume II, pages 69-88. Center for Security Studies, ETH Zurich, 2006.

[Winkler & Dealy 1995] I. S. Winkler and B. Dealy, “Information security technology?. . . don’t rely on it—a case study in social engineering”, Proceedings of the 5th USENIX UNIX Security Symposium, June 1995.

[Xu et al. 2003] J. Xu, Z. Kalbarczyk, and R. K. Iyer, “Transparent runtime randomization for security”, Proceedings of the 22nd Int.Symposium on Reliable Distributed Systems, pp. 260–269, October 2003.

[Zhou et al. 2002] L. Zhou, F. Schneider, and R. van Renesse, “A secure distributed on-line certification authority”, ACM Transactions on Computer Systems, 20(4), pp. 329–368, November 2002.

[Zhou et al. 2002b] L. Zhou, F. Schneider, and R. van Renesse, "COCA: A secure distributed on-line certification authority", ACM Transactions on Computer Systems, 20(4), pp. 329–368, November 2002.

[Zhou et al. 2005] L. Zhou, F. B. Schneider, and R. Van Renesse, "APSS: proactive secret sharing in asynchronous systems", ACM Transactions on Information and System Security, 8(3), pp. 259–286, 2005.

[Zielinski 2004] P. Zielinski, "Paxos at war", Technical Report UCAM-CL-TR-593, University of Cambridge Computer Laboratory, Cambridge, UK, June 2004.