The CRUTIAL Architecture for Critical Information Infrastructures ⋆

P. Verissimo, N. F. Neves, M. Correia
FCUL- U. Lisboa, Lisboa, Portugal

Y. Deswarte[1], A. Abou El Kalam[2]
Université de Toulouse
[1] LAAS-CNRS, [2] IRIT, ENSEEIHT-INPT
Toulouse, France

A. Bondavalli, A. Daidone
University of Florence
Florence, Italy

{pjv,nuno,mpc}@di.fc.ul.pt yves.deswarte@laas.fr
anas.abouelkalam@enseeiht.fr bondavalli@unifi.it
daidone@dsi.unifi.it

Abstract. In this chapter we discuss the susceptibility of critical information infrastructures to computer-borne attacks and faults, mainly due to their largely computerized nature, and to the pervasive interconnection of systems all over the world. We discuss how to overcome these problems and achieve resilience of critical information infrastructures, through adequate architectural constructs. The architecture we propose is generic and may come to be useful as a reference for modern critical information infrastructures. We discuss four main aspects: trusted components which induce prevention; middleware devices that achieve runtime automatic tolerance and protection; trustworthiness monitoring mechanisms detecting and adapting to non-predicted situations; organization-level security policies and access control models capable of securing global information flows.

1 Introduction

The largely computerized 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 this chapter, we are concerned with the susceptibility of the latter to computer-borne attacks and faults, i.e., with the protection of these infrastructures.

We propose an architecture and a set of techniques and algorithms aiming at achieving resilience to faults and attacks in an automatic way. Although we focus on the computer systems behind electrical utility infrastructures as an example, the architecture we propose is generic and may come to be useful as a reference for modern critical information infrastructures.

⋆ This work was mainly supported by the EC, through project IST-FP6-STREP 027513 (CRUTIAL) and NoE IST-4-026764-NOE (ReSIST), by the FCT through the Large-Scale Informatic Systems Laboratory (LaSIGE) and the CMU-Portugal partnership.

It is worthwhile recapitulating some of the reasoning behind the blueprint of this architecture, recently published [23]. Although inspired by previous intrusion-tolerant system architectures, the CRUTIAL architecture was largely influenced by two facts. Firstly, the fact that Critical Information Infrastructures (CII) feature a lot of legacy subsystems (controllers, sensors, actuators, etc.). Secondly, the fact that conventional security and protection techniques can bring serious problems, when directly applied to CII controlling devices, by preventing their effective operation. Although they are very practical problems, we will show ahead that they yielded in fact very interesting research challenges.

Another relevant fact was that our belief that the crucial problems in critical information infrastructures lie with the forest, not the trees, has been confirmed everyday as new incidents have occurred. That is, the problem is mostly created by the generic and non-structured network interconnection of CIIs, which bring several facets of exposure impossible to address at individual level. Whilst it seems today non-controversial that such a status quo brings a considerable level of threat, to our knowledge there had been no previous attempt at addressing the problem through the definition of a reference model of a *critical information infrastructure distributed systems architecture*. One which, by construction, would lay the basic foundations for the necessary global resilience against abnormal situations. Our conjecture was that such a model would be highly constructive, for it would form a structured framework for (1) conceiving the right balance between prevention and removal of vulnerabilities and attacks; (2) achieving tolerance of remaining potential intrusions and designed-in faults; and (3) enabling adaptation and self-awareness mechanisms to overcome unforeseen situations. In this chapter, we will report some advances in this area.

Finally, and in a related manner, we conjectured that any solution, to be effective, has to involve automatic control of macroscopic command and information flows, occurring essentially between the several realms composing the critical information infrastructure architecture (both intra- and inter-organizations), with the purpose of securing appropriate system-level properties, at organizational level. This has to be addressed, in an automatic way, through innovative access control models that understand the organizational reality, and are thus capable of translating the related high-level security policies into the adequate technical mechanisms such as access control matrices and firewall filter rule-sets.

The chapter is organized as follows: Section 2 does the Architecture Description. Then, the Protection Strategies and Services are introduced in Section 3, followed by the Trustworthiness Monitoring Services in Section 4. The chapter concludes with a discussion on Access Control for Critical Information Infrastructures, in Section 5.

2 Architecture Description

The CRUTIAL architecture encompasses four aspects. (i) Architectural configurations featuring trusted components in key places, which a priori induce prevention of some faults, and of certain attack and vulnerability combinations. (ii) Middleware devices that achieve runtime automatic tolerance of remaining faults and intrusions, supplying trusted services out of non-trustworthy components. (iii) Trustworthiness monitoring

mechanisms detecting situations not predicted and/or beyond assumptions made, and adaptation mechanisms to survive those situations. (iv) organization-level security policies and access control models capable of securing information flows with different criticality within/in/out of a CII. It is important to point out that the notion of CII is hard to formalize. The generic idea is that the CII is the computer systems (or ICT) part of a critical infrastructure, which is the working definition that we use in this chapter.

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 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. Middleware services and protocols in the architecture use distinct techniques that address different levels of criticality of the architecture, such as randomization and wormholes, software or hardware implementations, and support a diverse set of requirements from the applications: dynamic and static groups; synchronous, partially-synchronous, and asynchronous execution; tolerance from benign accidental faults to malicious coordinated attacks.

CRUTIAL Information Switches (CIS) route the information to and from LANs with different criticality levels, wherever they are in the infrastructure: intranet, SCADA, Internet gateway. In fact, a lot of the protection and intrusion resilience reside in this class of components that interconnect the several LANs comprising a CRUTIAL architecture. But they are more than mere TCP/IP routers: in a simplistic way they could be seen as sophisticated circuit or application level firewalls combined with equally sophisticated intrusion detectors, connected by distributed protocols. 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 system-level properties.

Monitoring and diagnosis can be performed at several levels, through diverse mechanisms: CIS self-diagnosis, the diagnosis inside the CIS as part of the fault tolerance policy of the CIS itself; diagnosis on other components in the system (making assumptions on the security policy applied inside the CIS); diagnosis on the LANs and their nodes; diagnostic information gained by processing the security policy decisions, interpreting them as error detections. The collected information may be used in order both to take local decisions and to coordinate CIS activities.

Access control is a key issue. Although several organizations are normally involved and have to cooperate in the operation of a CII, from the access control point of view, each organization in a CII should keep its independence and responsibility on its assets and personnel. We propose that: each organization defines its own security policy (according to the OrBAC model), and enforce it with its own authentication and authorization means; the organizations cooperate through web services, and for each web service, a contract is signed between the provider and the client; this contract is translated in security rules (expressed within the Poly-OrBAC model), these rules being implemented with the involved CIS, and enforced at each step of web service interaction; the interactions are recorded into logs by each involved CIS, and these logs can serve as evidence

in case of dispute: each organization stay liable of all actions initiated by its own personnel.

2.1 Key Architecture Aspects

The CRUTIAL architecture, despite inspired by previous intrusion-tolerant reference architectures like MAFTIA [22], extends them significantly to attend the specific challenges of the critical information infrastructure problem, for example, legacy, global access control, and above all non-stop operation and resilience.

Given the severity of threats expected, some key components are built using architectural hybridization methods in order to achieve extremely high robustness:

- *Trusted-trustworthy* operation [22] is 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. This concept is in line with, but richer than, recent technological concepts like trusted computing or trusted platform modules.

Another interesting aspect of this work is related with the mechanisms that we had to develop, to preserve the large legacy composition of CII and keep changes to a minimum:

- *Fully-transparent intrusion tolerance* aims at preserving the complete illusion of a standard system to legacy components. It is implemented by innovative replica control and communication algorithms. Any SCADA and corporate network technologies stay unchanged, the only modification foreseen being the requirement of IPsec at communication level, but this is considered a trend anyway [4].

Another innovative aspect of this work is our approach to achieve resilience. This goes further to mere intrusion tolerance, and can be seen as a specialization of this kind of architecture to critical infrastructures. The problem is addressed through two paradigms:

- *Proactive-resilience* to achieve exhaustion-safety [20], and ensure perpetual, non-stop operation despite the continuous production of faults and intrusions. This is not a requirement of many intrusion-tolerant systems, but it is definitely of importance for unattended operation, as is desired of the control part of CII.
- *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 [2]. In the latter case, dependable adaptation mechanisms are triggered to stabilize coverage and thus, the operational guarantees. This is of extreme importance for situations of instability, either caused by accidental events or malicious attacks, and we believe it can be a key to lower the risk of cascading and/or escalating failures.

Finally, the desired control of the information flows is partly performed through advanced protection mechanisms:

– the *OrBAC firewall* is an adaptation of the classical firewall rule-set operation to enforce an organization-based access control model (OrBAC) [11] for implementing global-level security policies. OrBAC allows the expression of security policy rules as high level abstractions, and it is of importance for homogenizing the diverse security policies of organizations involved in a CII into one policy that controls the global information flow.

In summary, 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.

2.2 Main Building Blocks

The overall picture of a CRUTIAL system, shown in Figure 1, was detailed in [23].

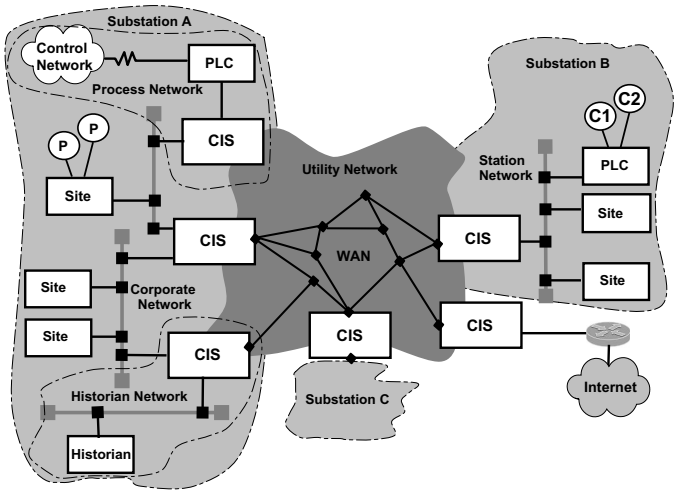

Fig. 1: CRUTIAL overall architecture (WAN-of-LANs connected by CIS)

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 *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. 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. Virtual LANs (VLANs)). More than one LAN can be connected by the same CIS. 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; an historian network (to store monitoring data); etc.

CIS collectively 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*. In consequence, no traffic set to enjoy CRUTIAL-level protection can go from one LAN to another without crossing a CIS.

3 Protection Strategies and Services

We now discuss the failure assumptions underpinning the architecture design, concerning the main architectural devices: WAN, LAN, CIS. Shadowing in the figure symbolizes untrusted areas:

- The WAN interconnect (heavily shadowed) is assumed to have arbitrary behavior, which is akin to saying it can be totally compromised.
- The CII facilities (lightly shadowed) are assumed to have varying faulty behavior, from arbitrary to crash failure.
- Inside the facilities, the LAN is the unit of failure. This is akin to assuming that some LANs will be completely trusted (e.g., by construction, or by recursive use of intrusion tolerance), whereas other LANs may even be arbitrary (e.g., in consequence of insider threats).
- Overall, we assume that faults (accidental, attacks, intrusions) continuously occur during the life-time of the system, the only limit being that a maximum number of f malicious (or arbitrary) faults can occur within a given interval. Note that this is weaker than assuming that only f faults may occur during the whole life-time of the system.
- CIS components are trusted to securely switch information flows as a service to edge LANs as clients.
- LANs trust the services provided by the CIS, but are not necessarily trusted by the latter.

The assumptions described above have a few implications on the protection strategies chosen. Let us start by the CIS construction:

- The CIS is a main target to any hacker having understood the CRUTIAL architecture, since the CIS is supposed to be a trusted component. We recognize this threat by assuming that a number of CIS or components thereof can be corrupted.
- In order to be trusted, the CIS must be trustworthy. As such, the CIS itself must be made intrusion-tolerant, prevent resource exhaustion providing perpetual operation (i.e., can not stop), and be resilient against assumption coverage uncertainty, providing survivability.

– The CIS is thus implemented as a set of redundant units (multiple-box physically replicated hardware units, or single-box logically replicated software units), depending on the level of resilience to attain.
– Given the nature of malicious faults, which can be made common-mode, CIS construction and or reconfiguration may be based on diversity techniques (ex. n-version programming, obfuscation, etc.).
– The CIS also has proactive recovery mechanisms, so that each component is periodically rejuvenated in such a way that if it suffered an intrusion, then the intrusion is no longer present after the rejuvenation process.
– The CIS is further monitored by special run-time trustworthiness monitoring mechanisms, which make sophisticated sanity checks. Reactive recovery may for example be triggered immediately an successful attack or failure is detected. Adaptation mechanisms may also be parameterized by these monitors.

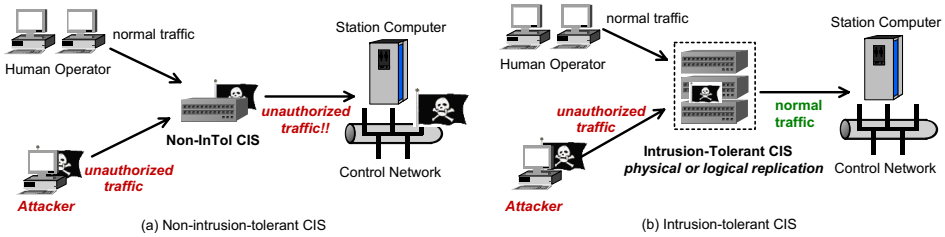

Fig. 2: Building trust in CRUTIAL (CIS level)

As exemplified in Figure 2a, a simplex (i.e., non intrusion-tolerant) CIS, once attacked, becomes under the control of the attacker and can fail. In the figure, attack traffic will go through the compromised CIS and hit the station computer and control networks in the CII. On the other hand, an intrusion-tolerant CIS (Figure 2b) despite corruption of one or more components, will continue providing correct service, as long as not more than a quorum f of component failures occur. CIS can be physically or logically replicated, examples of this incremental intrusion tolerance strategy are discussed in [1]. The figure shows a triple with a corrupted replica ($f = 1$): despite attacked by unauthorized traffic, voting between all replicas discards these messages, only letting normal traffic through to the station computer.

Let us look at the services running in or among CIS:

– The local services implemented on the CIS servers enjoy the CIS intrusion tolerance to secure the desired properties in the presence of malicious traffic and/or commands.
– The distributed services implemented on sets of CIS servers are subject to possibly Byzantine attacks. In consequence, cooperating CIS must be interconnected with intrusion-tolerant protocols, in order to correctly implement the desired services.

Consider that the CIS boxes in the next figures represent intrusion-tolerant logical CIS. That is, to any services running locally on top of a logical CIS, the latter appears as

being fail-controlled, in a good example of recursive use of intrusion-tolerance featured by this architecture. As exemplified in Figure 3a, services running on an intrusion-tolerant CIS are trusted to run correctly, despite faults or attacks. As such, these services are trusted-trustworthy, that is, trusted because they are trustworthy.

On the other hand, some services may need to be run cooperatively amongst CIS, and thus be subject to attacks at the WAN interconnect level. If one generalizes the distributed intrusion tolerance concept to CIS interconnection, one will run specialized intrusion-tolerant algorithms amongst CIS, which end-up achieving what is portrayed in Figure 3b: a trusted-trustworthy communication fabric amongst CIS, overcoming the initial untrusted basic WAN interconnect.

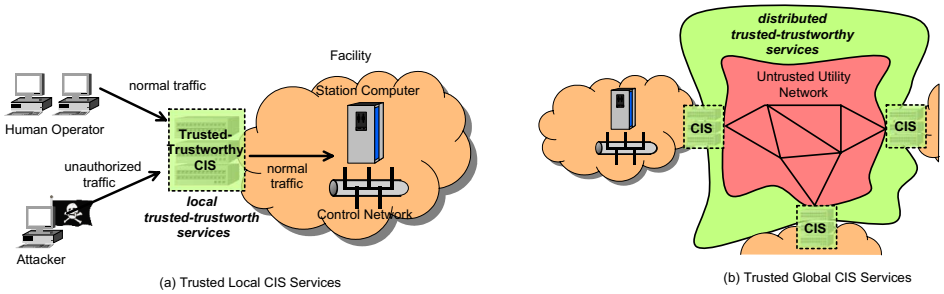

Fig. 3: Building trust in CRUTIAL (WAN level): a) Trusted Local CIS Services; b) Trusted Global CIS Services

4 Trustworthiness Monitoring Services

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/sub-systems used within the infrastructure already implement monitoring and/or recovery techniques that could in some way work "against" the needs of the infrastructure itself. It is hence necessary to coordinate all the monitoring and/or recovery activities in order to favor the infrastructure needs (sometimes despite of component/sub-system needs).
- 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 harbor the originating fault, as well as the specific commands able to bring each component

to a correct, consistent state; ii) otherwise, if 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 behavior 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 has to be 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 ones; it is not possible to manage these problems at the component level, but it is necessary to do that at middleware level (or at least at application 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 fault tolerance at application level.

4.1 The Diagnosis Framework

The diagnosis framework adopted to tackle these challenges involves the following actors (see Figure 4):

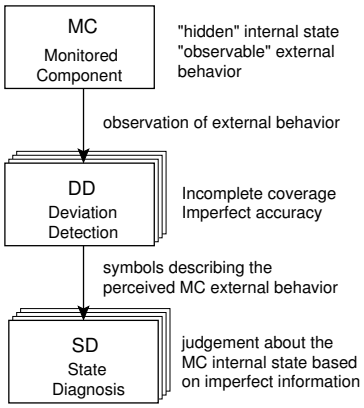

Fig. 4: The diagnosis framework

Monitored Component (MC). It 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 behavior. The internal healthy state[1] of the component is therefore "hidden", whilst the external behavior is "observable". Since the same observed incorrect behavior could be caused by different faults, it is an ambiguous indicator of the healthy internal health state of the component itself.

Deviation Detection (DD). It is the entity introduced in the system in order to observe the monitored component external behavior and to judge whether it is suitable or not. Unsuitable component behavior could be the result of the manifestation of an internal or external 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. In the literature on fault tolerance, a wide variety of error detection mechanisms are available, which are classified in different categories according to several criteria, among which the type of checks they perform, the implementation support (hw or sw), the system components they are tailored to, the applicability time (on-line or off-line) [19]. 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 judgment describing the instantaneous external behavior 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 it, whilst in the reactive schema the entity that generates information has to send it to the entity interested in it. More interaction patterns can be found in [17].

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

[1] 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.

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.

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[2] is performed. Literature shows that many over-time diagnostic mechanisms are available; each approach can be mapped on the schema presented earlier 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 models. Most heuristic diagnosis solution are based on a count-and-threshold approach [14], as exemplified by alpha-count [3] a heuristic designed to discriminate whether a 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 consists 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:

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

Every time the counter is evaluated, it is also compared with a predefined threshold value α_t in order to discriminate if an alarm has to be raised ($\alpha(t) \geq \alpha_t$) or not ($\alpha(t) < \alpha_t$). Extended analyzes about parameter tuning are available in [3] while applications and variants are described in [2] [18].

Probabilistic diagnosis. Probabilistic diagnosis mechanisms (e.g. [6] based on HMM - Hidden Markov Models and [16] based on Bayesian inference) are tailored to evaluate the probabilities of the monitored component being in each of the "internal" state envisioned in the fault model, based on symbols coming from the deviation detection mechanism. Probabilistic diagnostic mechanisms compute a state occupancy probability vector $f(t)$ at time t, using 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

[2] A syndrome is a vector of Boolean deviation detection results.

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 all the possible outcomes of some experiment. Using the Bayes' theorem we can write that in general:

$$p(x\,|\,evidence)\cdot p(evidence) = p(evidence\,|\,x)\cdot p(x)$$

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

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

Given a set C of mutually exclusive conjectures such that their union has probability 1 (e.g. healthy 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:

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

Both approaches solve the problem with the same computational cost, but diagnosis based on HMM 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).

Diagnosis activity has to be performed at different granularity levels (Fault Replacement Unit), depending on the controllability of control on 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 in which every 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 [28] 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 sub-system 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, aiming to diagnose the sub-system itself.
2. Global information collection and global diagnosis, obtained through exchange of local diagnosis. Since each sub-system may have a different perception of the errors observed on the remote sub-systems, each node has some private values (the results of private diagnosis on remote sub-systems) and the goal is to ensure consistent information exchange and agreement against Byzantine behavior. An agreement (or consensus) algorithm is thus needed in order to solve the problem.

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 [8], which can be copied and forwarded but not altered without detection, the condition on the minimal number of nodes can be relaxed to $f+2$.

4.2 Diagnosis in CRUTIAL

The CRUTIAL infrastructure is organized as a WAN-of-LANs (see Section 2.2), where each LAN is connected to the WAN by a CIS. Given that the computers inside the LANs cannot be modified/updated, all the diagnosis activity has to be performed inside the CIS. The following diagnosis scenarios arise:

- CIS self-diagnosis (local view): CIS monitors both itself (e.g. to diagnose hardware or software faults) and its LAN (e.g. to "measure" its level of trustworthiness).
- CIS distributed diagnosis (global view): CISes construct a common view about the "state" of a certain CIS in the infrastructure (e.g. related to the liveness and trustworthiness of a specific CIS)

CIS self-diagnosis From a local viewpoint the CIS is a sophisticated application level firewall (combined with equally sophisticated intrusion detectors) which is required to:

- be intrusion-tolerant;
- prevent resource exhaustion providing perpetual operation;
- be resilient against fault assumption coverage uncertainty providing survivability.

In order to comply with the above requirements, the CIS has a hybrid architecture and is replicated (with diversity) in n replicas. Each CIS replica is built using a synchronous and secure local wormhole and an asynchronous and insecure payload.

Two monitoring/failure detection scenarios arise:

1. internal monitoring: monitoring performed inside a single replica, trying to detect local failures;
2. external monitoring: monitoring performed on the perceived behavior of the other replicas.

The internal monitoring has to be performed on the following components/services (so far, components/services that need to be monitored were not definitely identified):

- Hardware components (e.g. network interfaces, processing units, memory modules...) which are supporting the replica. The monitoring activity on these components makes sense only when physical replication is used; in case of logical replication, these components need to be monitored in the host system running the replicas.
- Software components belonging to several architectural levels in the payload or in the operating system.

Several signals coming from many architectural levels are collected and processed over time: an example of signal coming from low architectural levels (O.S.) is related to a CPU fan that is working too slow or a temperature sensor that is signaling the CPU is too warm. An example of signal coming from a higher architectural level is an application-generated exceptions or error return code.

The internal monitoring activity has hence to identify compound system conditions which could require diverse corrective actions; for example, repeated application errors could be interpreted as manifestation of software aging requiring rejuvenation, or could be correlated with lower level signals (the CPU is too warm because the CPU fan is working too slow), requiring another kind of reconfiguration (e.g. replacing the CPU fan). The rationale behind internal monitoring and failure detection is to try to stop the replica before it starts to behave incorrectly.

The external monitoring is performed by each replica on the perceived behavior of the other replicas, given that a replica is not guaranteed to always behave correctly. The monitoring activity is performed at service level, so that each service is in charge of detecting whether its peers running in the other replicas seem correct or not. An example of middleware service monitoring its peers on other replicas is the Protection Service.

CIS LAN Diagnosis The CIS monitors over time the nodes in its protected LAN to evaluate their trustworthiness. The evaluated trustworthiness level is used to request maintenance actions on the protected node (e.g. replacing hardware, refreshing the software, changing passwords...).

A trustworthiness indicator for each protected node N is defined (it could be multi-dimensional) and modified based on the following events:

- the instance of the security policy applied within the CIS itself to the outgoing traffic detects that N is trying to violate the security policy (e.g. trying to send something without being allowed to do it);
- the instance of the security policy running on a remote CIS detects that a message sent by N to one of its protected nodes was rejected. The CIS distinguishes whether an incoming packet really comes from a station computer (instead from an hacker in the WAN) using the LAN Traffic Labeling service (the CIS protecting the source node signs the label). The signed label is hence a proof of the source of the packet.

The LAN Diagnosis service collects over time the above detections in order to evaluate the trustworthiness indicator of each protected node. If protected trustworthiness indicator of node *N* goes over a given threshold, the LAN Diagnosis service alerts its peers about *N* being un-trustable (so that they can possibly take adequate countermeasures).

CIS Distributed Diagnosis The several replicas that made up a single CIS are required to perform the same operations; this simplifies somewhat the task of checking their correctness on the run. Each single CIS, as seen from the WAN, is a different logical entity, in terms of actions, services and requests toward other CISes. In the ordinary information flow there is no simple comparison rule check that can be performed, to catch on the fly a mischievous partner. On the other hand, if a CIS becomes compromised, internal redundancy and resilient architecture notwithstanding, then necessarily the basic hypothesis on the fault occurrence has been broken: more than f replicas are out of order together. Of course, this is the catastrophic case, whose probability has to be lowered down to a target level by choosing proper redundancy figures. However, a local catastrophe (regarding a single LAN controlled by a compromised CIS) not necessarily should imply the downing of the entire system. In fact, on the WAN side, all CISes attempt to maintain a common view of two parametric descriptors its partners' health: Liveness and Trustworthiness.

Liveness is checked in two ways: i) passively, by monitoring normal network traffic from the target; ii) if the former is not frequent enough, exert a form of resilient ping, by means of a simple challenge/response protocol. Trustworthiness is built up by checking the formal correctness of the messages coming from the target, as well from any access violation detected by the Protection Service.

5 Access Control for Critical Information Infrastructures

Because Critical Information Infrastructures (CII) become more and more complex and accessible via the Internet, they are more and more vulnerable to security threats. Moreover, due to the interdependencies between CIIs, simple failures can have dramatic consequences. In this context, security issues in CIIs become obvious and serious. Several works was dedicated to study the causes; the results have shown that one of the most common problems is the lack of specific security policies, in particular in modern SCADA environments [12].

Basically, a security policy is defined by the Common Criteria as *the set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information* [15]. In this respect, a security policy is specified through: the security *objectives* that must be satisfied, e.g., "*classified information must not be disclosed to a competing organization*"; and the *rules* expressing how the system may evolve in a secure way, e.g., "*the owner of an information is allowed to grant a read access right on its data to other organizations*".

Nevertheless, by itself, the security policy does not guarantee a secure and correct functioning of the system. The security policy can indeed be badly designed or intentionally / accidentally violated. Consequently, it is important to express the policy according to a security model; a model helps to: abstract the policy and handle its

complexity; represent the secure states of a system (i.e., states that satisfies the security objectives) as well as the way in which the system may evolve; verify the consistency of the security policy and detect the possible conflicting situations; etc.

Addressing these issues, this work progressively derives an access control model, a secure architecture and applies our approach to secure CIIs. To do so, we first identify the security requirements of a CII and we confront them to existing access control models. Note that even if we take our examples from electric power grid, the same approach and results can be applied to any kind of CII.

Globally, a CII can be seen as a WAN connecting several organizations involving different actors and stakeholders (e.g., power generation companies, substations, energy authorities, maintenance service providers, transmission and distribution system operators) and various LANs. LANs are composed of one or more logical and physical systems and are interconnected through specific Switches, called CIS (*CRUTIAL Information Switches*). The general architecture is presented in Section 2.

In this respect, we can identify some security-related requirements such as:

1. *Secure cooperation* between different organizations, possibly mutually suspicious, with different features, functioning rules and policies.
2. *Loosely coupled organizations*: each organization controls its own security policy, applications, etc., while respecting the global functioning of the whole system. In other words, we need a global security policy that manages the communication between partner organizations while keeping each CII responsible for its own assets and users.
3. *Coherence and consistency*: as no SCADA system operates in isolation, the global as well as local security policies should be compatible.
4. *Decentralization*: it is desirable that the enforcement and administration of the security policies be decentralized. Actually, a centralized approach is not interesting since CIIs involve the cooperation between independent organizations. Inversely, handling the collaboration between the organization subsystems while keeping some local self-determination seems more interesting.
5. *Heterogeneity*: the different CII organizations have their own structure, services, OS, and local objects. These entities' structures may be different from an organization to another.
6. *Granularity vs. scalability*: on the one hand, security rules must be extensible in size, structure, and number of organizations; on the other hand, internal authentication as well as local access controls should be managed by each organization separately.
7. *Fine-grained access control*: access decisions should take the context (e.g., specific situations, time and location constraints) into account. Moreover, as the context may change often and as certain reactivity is required in these systems, organizations should support dynamic access rights.
8. *Users-friendliness and easiness of rules administration*: as the global system links several organizations geographically distributed and as it handles a large amount of information and a big number of user, the access right management should be sufficiently user-friendly to manage this complexity without introducing errors.

9. *Remote accesses*: as organizations control large installations, the security policy should define if and how outsiders and users from a partner organization can connect to the automation system and to resources belonging to each organization. For example, it is important to define how vendors can access the system remotely for off-site maintenance and product upgrades, but also how other organizations participating in the CII can access local resources.
10. *Compliance with the specific regulation*: for example, in the United-States, NERC 1200 [5] specifies requirements for cyber-security related to electric utilities.
11. *Confidentiality, integrity and availability*: contrarily to other systems where mostly confidentiality (military systems), integrity (financial systems) or availability is needed, in the organization we often need these three properties (confidentiality of each organization's data, e.g., invitation of tenders, but also integrity and availability of data such as the voltage/frequency measurements).
12. *Enforcement of permissions, explicit interdictions as well as obligation rules*. In fact, explicit prohibitions can be particularly useful as we have decentralized policies where each administrator does not have details about the other parts of the system. Moreover, explicit prohibitions can also specify exceptions or limits the propagation of permissions in case of hierarchies. Similarly, obligations can be useful to impose some internal / external, manual / automatic actions that should be carried out by users or automatically by the system itself.
13. *The security policy must be vendor- and manufacturer-independent*. As technologies change and new acquisitions occur, the policy must remain effective. When vendor- or technology-specific statements are used, the maintenance burden for the policy increases. Then, the policy would be changed any time there is a new purchase or an advance in technology. If the security policy is not updated, it becomes obsolete, which would not be acceptable in such systems.
14. *Audit and assessment*: the security policy will define the logging requirements such as what will be logged, when, where, etc. In particular, an audit will determine if the protections which are detailed in the policy are being correctly put in practice on the system; it also checks if the contracts / agreements established by the partner organizations are well-respected.

To satisfy these requirements, we propose a secure architecture where each organization defines its own security policy and enforces it into its CISs (see Section 2). CISs are thus responsible for checking if local actions are in accordance with internal security policies, but also if inter-organization flows are done according to the global policy and to the contracts established by partner organization. Finally, CISs keep log files as evidence in case of abuse or conflict. In this respect, the first question that arises is how the security policies will be specified and what security model will be adapted to organizations?

An analysis of classical access control policies and models shows that unfortunately, none of these policies and models satisfies the CIIs security-related requirements. For instance, HRU represents the relationships between the subjects, the objects and the actions by a matrix M [9]. $M(s,o)$ represents the action that 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 by the policy, which is very complex in large systems. Moreover, when new en-

tities are added to or removed from the system, it is necessary to update the policy, still adding to the complexity. Consequently, models associated to discretionary access control policies (including HRU) are not capable of managing huge, multi-organizational and decentralized systems such as CIIs.

Role Based-Access Control (RBAC) is more flexible: roles are assigned to users, permissions are assigned to roles and users acquire permissions by playing roles [7]. Even if RBAC is suitable for a large range of organizations, it does not cover all the requirements of a CII, in particular it does not define how users of an organization can play roles in another organization.

5.1 OrBAC

In [11] we have defined the OrBAC (*Organization-based Access Control*) model as an extension of RBAC that details permissions while remaining implementation independent. Our first goal was to express the security policy with abstract entities only, and thus to separate the representation of the security policy from its implementation. Indeed, OrBAC is based on roles as the abstraction of users (like in RBAC [7]), views as the abstraction of objects (like in VBAC [24]), and activities as the abstraction of actions (like in TBAC [21]).

Actually, 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.

As a user can play several roles in several organizations, the *Role* entity is used to structure the link between the subjects and the organizations. In fact, contrarily to RBAC that considers a binary relation between roles and subjects, OrBAC consider the ternary relationship *Empower (org, r, s)*: it means that *org* employs subject *s* in role *r*. In the same way, the objects that satisfy a common property in a certain organization are specified through views (the *Use*(*org, view, object*) relationship), and activities are used to abstract actions in organizations (the *Consider* (*org, activity, action*) relationship) (Figure 5).

Now, once the relationships between the different system' entities are defined, we can specify the security rules. Actually, an OrBAC security rules have the *Permission* (*org, r, v, a, c*) form: in the context *c*, organization *org* grants role *r* the permission to perform activity *a* on view *v*. *Obligation* and *Prohibition* are defined similarly (*Obligation* (*org; r; v; a; c*) and *Prohibition* (*org, r, v, a, c*)).

Actually, two security levels can be distinguished in OrBAC (Figure 5):

- *Abstract level*: the security administrator defines security rules through abstract entities (roles, activities, views) without worrying about how each 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 (e.g., the context).

The derivation of permissions (i.e., runtime instantiation of security rules) can be formally expressed as indicated in (Figure 5):

$\forall org \in Organization, \forall s \in Subject, \forall \alpha \in Action, \forall o \in Object, \forall r \in Role, \forall a \in Activity,$	
$\forall v \in View, \forall c \in Context$	
Permission(org, r, v, a, c) $\wedge$	// **a security rule in its abstract form**
Empower(org, s, r) $\wedge$	// in *org*, the role *r* is played by a subject *s*
Consider(org, α*, a)* $\wedge$	// in *org*, the activity *a* correspond to an action α
Use(org, o, v) $\wedge$	// in *org*, the view *w* corresponds to an object *o*
Hold(org, s, α*, o, c)*	// in *org*, the context *c* is true for *s*, α and *o*
$\rightarrow$ *is-permitted(s,* α*, o)*	// **runtime decision allowing** *s* **carrying out** α **on** *o*

This rule means: *if* in a certain organization, a security rule specifies that "role *r* can carry out the activity *a* on the *v* when the context *c* is True"; *if* "*r* is assigned to subject *s*"; *if* "action α is a part of *a*"; *if* "object *o* is part of *v*" and, *if* "the context *c* is True; *Then s* is allowed to carry out α on *o*.

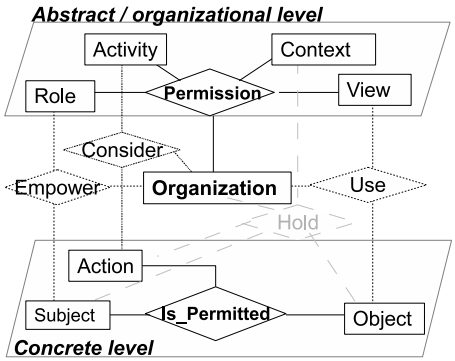

Fig. 5: The ORBAC model.

As rules are expressed only through abstract entities, OrBAC is able to specify the security policies of several collaborating and heterogeneous organizations (e.g., in a CII), if they are considered as "sub-organizations" of a "global organization" with a single OrBAC policy. In fact, the same role, e.g., "*operator*", can be played by several users belonging to different sub-organizations; the same view, e.g., "*TechnicalFile*", can designate a *TF-Table* or a *TF1.xml*; and the same activity "*read*" could correspond in a particular sub-organization to a "*SELECT*" action (if the sub-organization has a database system) while in another sub-organization it may specify an *OpenXMLfile()* action.

In our context, OrBAC present several benefits and satisfies several security requirements of CIIs:

- *Rules expressiveness:* OrBAC defines permissions, interdictions, obligations, and constraints (by means of contextual conditions).
- *Abstraction of the security policy:* OrBAC has a structured expression of the policy; it separates the specification of the policy from its implementation. Consequently, OrBAC greatly reduces the cost of administering security policies as well as making the process less error-prone.

- *Scalability:* thanks to its abstraction levels, OrBAC has no limitation in size or capacity. It can define an extensible and huge policy. It is then easily applicable to large-scale environments.
- *Loose coupling:* each sub-system can manage its own local OrBAC security policy, as far as it respects the global policy.
- *Evolvable:* a local policy in OrBAC is evolvable. It easily handles changes in organizations.
- *User-friendly:* the specification and update of a local OrBAC security policy is easily managed at the local organization level.
- *Standardized:* OrBAC has a growing community. Many research tracks are being conducted (see www.orbac.org).

However, OrBAC is centralized and does not handle collaborations between organizations, while these aspects are very important in CIIs. In fact, as OrBAC security rules have the *Permission(org, r, v, a, c)* form, it is not possible to represent rules that involve several independent organizations, or even, autonomous sub-organizations of a particular collaborative system. Moreover, it is impossible (for the same reason) to associate permissions to users belonging to other partner-organizations. As a result, if we can assume that OrBAC provides a framework for expressing the security policies of several organizations, it is unfortunately only adapted to centralized structures and does not cover the distribution, collaboration and interoperability needs of current CIIs.

Moreover, the enforcement of the policy to access control mechanisms is not treated in OrBAC. It is thus necessary to describe suitable architecture and implementation of the studied system's security.

To cover these limitations, we suggest enhancing OrBAC with new concepts and calling on some mechanisms of the Web Services (WS) technology [10]. In fact, WS is a set of technologies that provide platform-independent protocols and standards used for exchanging heterogeneous interoperable data services. Software applications written in various programming languages and running on various platforms can use WS to exchange data over computer networks in a manner similar to inter-process communication on a single computer. WS also provide a common infrastructure and services (e.g., middleware) for data access, integration, provisioning, cataloging and security. These functionalities are made possible through the use of open standards, such as: XML for exchanging heterogeneous data in a common information format [26]; SOAP acts as a data transport mechanism to send data between applications in one or several operating systems [25]; WSDL is used to describe the services that an organization (e.g., a CII) offers and to provide a way for individuals and other organizations to access those services [27]; UDDI 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 [13].

Web services (WS) have several benefits that could be interesting in our context:

- *Interoperability and heterogeneity*: WS support data exchanges between different platforms.
- *Resources sharing*: WS are adapted to applications where organizations share their resources.

- *Standardized mechanisms*: WS use open protocols and standards (e.g., HTTP, XML).
- *Easiness*: a small amount of code and resources is necessary to develop and carry out a WS.
- *Compatibility with OrBAC*: it is easy to couple web services with OrBAC.

5.2 PolyOrBAC

At this stage, we have demonstrated that OrBAC as well as WS could be suitable for CIIs. The question that takes place is: how adapting OrBAC as well as WS mechanisms to specify and enforce secure collaboration between CIIs. To answer this question, we have defined the PolyOrBAC [10], a global access control model that can be perfectly applied to CIIs.

Actually, PolyOrBAC distinguishes two phases:

First phase *publication and negotiation of collaboration rules as well as the corresponding access control rules.* First, each organization determines which resources it will offer to external partners. Web services are then developed on application servers, and referenced on the Web Interface to be accessible to external users.

Second, when an organization publishes its WS at the UDDI registry, the other organizations can contact it to express their wish to use the WS. To highlight the PolyOrBAC concepts, let us take a simple example where organization *B* offers *WS*1, and organization *A* is interested in using *WS*1 (Figure 6).

Third, *A* and *B* negotiate and come to an agreement concerning the use of *WS*1. Then, *A* and *B* establish a contract and jointly define security rules concerning the access to *WS*1. These security rules are registered – according to an OrBAC format – in a database (connected to the *A* and *B*'s CIS) containing the Security policy. For instance, if the agreement between *A* and *B* is "*users from A have the permission to consult B's measurements in the emergency context"*, *B* should:

- have (or create) a rule that grants the permission to a certain role (e.g., *Operator*) to consult its measurements: *Permission*(*B, Operator, Measurements, Consulting, Emergency*); note that every user playing the Operator role will have this permission
- create a "virtual user" *PartnerA* that represents *A* for its use of *WS*1
- add the *Empower(B, PartnerA, Operator*) association to its rule base.

In parallel, *A* creates locally a "virtual object" *WS*1_*image* which represents *WS*1, and adds a rule in its OrBAC base to define which of *A*'s roles can invoke *WS*1_*image* to use *WS*1.

Second phase: *runtime access to remote services*.

Let us first precise that we use an *AAA* (*Authentication, Authorization and Accounting*) architecture: we separate authentication from authorization; we distinguish access control decision from permissions enforcement; and we keep access logs in CISs. Basically, if a user from *A* (let us note it *Alice*) wants to carry out an activity, *she* is first authenticated by *A*. Then, protection mechanisms of *A* check if the OrBAC security policy (of *A*) allows this activity. We suppose that this activity contains local as well as external accesses (e.g., invocation of *B*'s *WS*1). Local accesses should be controlled

according to *A*'s policy, while the *WS*1's invocation is both controlled by *A*'s policy (Alice must play a role that is permitted to invoke *WS*1_*image*), and by *B*'s CIS, according to the contract established between *A* and *B*. If both controls grant the invocation, the execution of *WS*1 is executed under the control of *B*'s OrBAC policy (in *B*, *PartnerA* plays role *Operator* that is permitted to consult measurements).

More precisely, when *Alice* is authenticated and authorized (by *A*'s policy) to invoke *WS*1, an XML-based authorization ticket "$T1$" is generated (based on the positive decision) and granted to *Alice*.

$T1$ contains the access-related information such as: the virtual user played by *Alice*: "*PartnerA*"; *Alice*'s organization: "*A*"; the contract ID; the requested service: "*WS*1"; the invoked method, e.g., "*Select*"; and a timestamp to prevent replay attacks.

Note that $T1$ is delivered to any user (from A) allowed to access to *WS*1 (e.g., *Jean, Alice*). When *Alice* presents its request as well as $T1$ (as a proof) to *B*, *B*'s CIS extracts the $T1$'s parameters, and processes the request. By consulting its security rules, *B* associates the *Operator* role to the virtual user "*PartnerA*" according to *Empower*(*B, PartnerA, Operator*)[3]. Finally, the access decision is done according to *Permission*(*B, Operator, Measurements, Consulting, Emergency*) $\wedge$*gmpower(B, PartnerA, Operator*).

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*: all security rules are specified in OrBAC independently by each organization, and the organizations remain loosely coupled (through jointly agreed rules, expressed by contracts).
- *Information non-disclosure*: the WS technology allows communications between organizations without intimate knowledge of each other's IT systems; moreover, even if remote accesses are possible, it is not necessary to know the internal structure of the other organizations.
- *Extensible structure*: the OrBAC extensibility and the WS standards facilitate the management and the integration of new organizations (with their users, data, services, policy, etc.).

5.3 A scenario

Let us now apply PolyOrBAC to a real electric power grid scenario: in emergency conditions, the TS CC (*Transmission System Control Center*) can trigger load shedding on the DS (*Distribution System*) to activate defense plan actions (e.g., load shedding activities) on the Distribution Grid. More precisely, the TS CC (*Transmission System Control Center*) monitors the Electric Power System and elaborates some potentially emergency conditions that could be remedied with opportune load shedding commands applied to particular areas of the Grid.

[3] Let us recall that *Empower*(*B, PartnerA, Operator*) has been added after the negotiation phase (phase 1).

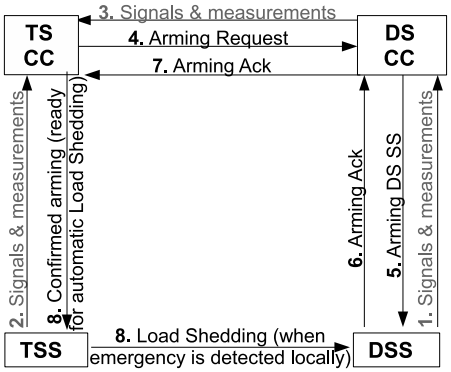

Fig. 6: The exchanged commands and signals.

Actually, as indicated in Figure 6, during normal operation, the Distribution Substations (DSS) send signals and measurements (voltage, Frequency, etc.) to the Transmission System Control Center TS CC (via the Distribution System Control Center DS CC); in the same way, the Transmission Substations (TSS) send signals and measurements (voltage, frequency, etc.) to the TS CC (steps 1, 2 and 3 in Figure 6).

At the TS CC level, when the TSO (*Transmission System Operator*) detects that a load shedding may be needed in the near future, it sends an arming request to the Distribution System CC (step 4 in Figure 6).

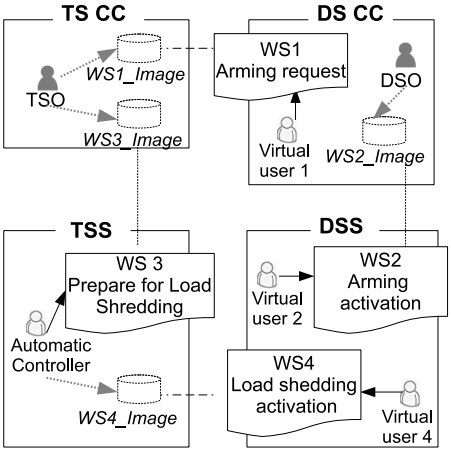

Fig. 7: The different WS invocations.

Consequently, the DSO (*Distribution System Operator*) selects which distribution substations (DSS) must be armed (these substations are those on which the load shed-

ding will apply if a load shedding is necessary), and then sends arming commands to those DSS. The DSO has naturally the permission to arm or disarm any DSS in the depending area of the DS CC.

If a Transmission SS (TSS) detects an emergency, it triggers (sends) a load shedding command to all the distribution substations (DSS) of its area. Of course, only the DSS already armed will execute the load shedding command.

In this scenario, we distinguish four organizations (TS CC, a TSS, DS CC and a DSS), two roles (TSO and DSO) and four web services (Figure 7): *Arming Request*, *Arming Activation*, *Confirmed Arming* and *Load Shedding Activation*.

WS1-Arming Request	Provider: DS CC Client: a user (TSO) or a process at TS CC
WS2-Arming Activation	Provider: DSS Client: DSO or a virtual user at DS CC
WS3-Confirmed Arming	Provider: TSS Client: a virtual user at TS CC
WS4-Load Shedding Activation	Provider: DSS Client: a user (automatic controller) at TSS

Table 1: The different WS of our scenario.

Basically, when negotiating the provision/use of *WS*1 between TS CC and DS CC, on the one hand, TS CC locally stores the WSDL description file and creates a new object as a local image of WS1 (whose actions correspond to WS1 invocations), and on the other hand, DS CC creates a virtual user (playing a role authorized to invoke *WS*1) to represent TS CC.

Moreover, TS CC adds local rules allowing Alice, a user playing the role TSO, to invoke *WS*1_*image*: *Empower(TS CC, Alice, TSO)*, and *Permission*(*TS CC, TSO, Access, TSO Distribution Circuits, Emergency*). In this respect, when Alice requests the access to *WS*1, the access decision is done according to the following rule:

Permission(TS CC, TSO, Access, TSO Distribution Circuits, Emergency) $\wedge$
Empower(TS CC, Alice, TSO) $\wedge$
Consider(TS CC, rwx, Access) $\wedge$
Use(TS CC, WS1-Image, TSO Distribution Circuits) $\wedge$
Hold(TS CC, Alice, rwx, WS1_Image, emergency) $\wedge$
$\rightarrow$ *is-permitted(Alice, rwx, WS1_Image)*

Besides, at the DS CC side, two rules are added: *Empower(DS CC, Virtual_User1, Operator)* and *Permission(DS CC, Operator, Access, DSO Distribution Circuits, emergency)*. Consequently, when Alice invokes WS1_Image, this invocation is transmitted to the DS CC by activating a process (running for Virtual_User1) which invokes *WS*1. This access is checked according to DS CC's policy and is granted according to the rule:

Permission(DS CC, Operator, Access, DSO Distribution Circuits, Emergency) $\wedge$
Empower(DS CC, Virtual_User1, Operator) $\wedge$
Consider(DS CC, rwx, Access) $\wedge$
Use(DS CC, WS1, TSO Distribution Circuits) $\wedge$
Hold(DS CC, Virtual_User1, rwx, WS1, emergency) $\wedge$
$\rightarrow$ *is-permitted(Alice, Virtual_User1, WS1)*

The other Web Services are negotiated and activated in the same way. This example shows that PolyOrBAC is a convenient framework for the security of Critical Information Infrastructures.

6 Conclusion

The chapter presented a distributed systems architecture for resilient critical information infrastructures, with respect to both accidental faults and malicious attacks and intrusions. Several aspects, such as design decisions and innovative mechanisms, were discussed and explained, in order to guide the reader through the making of such architectures.

The rationale for this work was based on three fundamental propositions: classical security and/or safety techniques alone will not be enough to solve the problem; any effective solution has to involve automatic control of macroscopic command and information flows between the LANs composing the CII; and, the unifying paradigm should be a reference architecture of "resilient critical information infrastructures" performing the integration of the different realms of a CII system.

The proposed solution encompasses a range of mechanisms of incremental effectiveness, to address from the lower to the highest criticality operations in a CII. Architectural configurations with trusted components in key places induce prevention of some attacks. Middleware software attains automatic tolerance of the remaining faults and intrusions. Trustworthiness enforcing and monitoring mechanisms allow unforeseen adaptation to extremely critical, not predicted situations, beyond the initial assumptions made.

Functionally, the information flow is controlled by basic mechanisms of the firewall and intrusion detection type, complemented and parameterized by organization-level security policies and access control models, capable of securing information flows with different criticality within a CII and in/out of it.

Some of the services running in CIS may require some degree of timeliness, given that SCADA implies synchrony, and this is a hard problem with malicious faults, so we plan to do research in this issue. We also take into account that these systems should operate non-stop, a hard problem with resource exhaustion, since the continued production of faults during the life-time of a perpetual execution system leads to the inevitable exhaustion of the quorum of nodes needed for correct operation.

Acknowledgements

CRUTIAL is a project of the IST programme of the European Commission. Several institutions participate to the project: CESI RICERCA (Italy), FCUL (Portugal), CNR-ISTI (Italy), LAAS-CNRS (France), K.U.Leuven-ELECTA (Belgium), CNIT (Italy).

Details about the project can be found at: `http://crutial.cesiricerca.it/`. We warmly thank our partners at the project for many discussions on the topics of the chapter. We also thank our colleagues and students in our research groups for their collaboration and feedback on this work.

References

1. A. N. Bessani, P. Sousa, M. Correia, N. F. Neves, and P. Verissimo. Intrusion-tolerant protection for critical infrastructures. DI/FCUL TR 07–8, Department of Informatics, University of Lisbon, April 2007.
2. 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):223–237, 2004.
3. 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):230–245, 2000.
4. 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. Revision 1.4.
5. North American Electric Reliability Council. Urgent action standard 1200, 2003.
6. A. Daidone, F. Di Giandomenico, A. Bondavalli, and S. Chiaradonna. Hidden Markov models as a support for diagnosis: Formalization of the problem and synthesis of the solution. In *25th IEEE Symposium on Reliable Distributed Systems (SRDS 2006)*, pages 245–256, October 2006.
7. D. Ferraiolo, R. Sandhu, S. Gavrila, D.Kuhn, and R. Chandramouli. A proposed standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3), 2001.
8. L. Gong, P. Lincoln, and J. Rushby. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults. *Dependable Computing for Critical Applications, IFIP WG 10.4, preliminary proceedings*, 5:79–90, 1995.
9. M.A. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August 1976.
10. A. A. El Kalam, Y. Deswarte, A. Baina, and M. Kaaniche. Access control for collaborative systems: A web services based approach. In *Proceedings of the IEEE International Conference on Web Services*, pages 1064–1071, 2007.
11. A. A. El Kalam, R. Elbaida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization-based access control. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, pages 277–288, June 2003.
12. D. Kilman and J. Stamp. Framework for SCADA security policy. Technical report, Sandia Corporation, 2005.
13. J. H. Lala, editor. *Foundations of Intrusion Tolerant Systems*. IEEE Computer Society Press, 2003.
14. G. Mongardi. Dependable computing for railway control systems. In *Proceedings of the International Conference on Dependable Computing for Critical Applications*, pages 255–277, 1993.
15. International Standards Organization. ISO/IEC Standard 15408-1, Common Criteria for Information Technology Security Evaluation, v3, Part 1: Introduction and general model, July 2005.

16. M. Pizza, L. Strigini, A. Bondavalli, and F. Di Giandomenico. Optimal discrimination between transient and permanent faults. In *Proceedings of the 3rd IEEE High Assurance System Engineering Symposium*, pages 214–223, 1998.
17. L. Romano, A. Bondavalli, S. Chiaradonna, and D. Cotroneo. Implementation of threshold-based diagnostic mechanisms for COTS-based applications. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, pages 296–303, October 13-16 2002.
18. M. Serafini, A. Bondavalli, and N. Suri. Online diagnosis and recovery: On the choice and impact of tuning parameters. *IEEE Transactions on Dependable and Secure Computing*, 4(4):295–312, 2007.
19. D.P. Siewiorek and R.S Swartz. *Reliable Computer Systems: Design and Evaluation*. A.K. Peters, 1998.
20. P. Sousa, N. F. Neves, and P. Verissimo. How resilient are distributed f fault/intrusion-tolerant systems? In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, June 2005.
21. R. Thomas and R. Sandhu. Task-based authorization controls. In *Proceedings of the 11th IFIP Working Conference on Database Security*, pages 166–181, 1997.
22. P. Verissimo, 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):54–62, Jul./Aug. 2006.
23. P. Verissimo, N. F. Neves, and M. Correia. The CRUTIAL reference critical information infrastructure architecture: A blueprint. *International Journal of System of Systems Engineering*, to appear 2008.
24. J. Vitek and C. Jensen. A view-based access control model for CORBA. In *Secure Internet Programming*, volume 1603 of *LNCS*. Springer, 1999.
25. W3C. SOAP, version 1.2. W3C Recommendation, June 2003.
26. W3C. Extensible markup language (XML). W3C Recommendation, February 2004.
27. W3C. WSDL, version 2.0. W3C Candidate Recommendation, March 2006.
28. Chris J. Walter, Patrick Lincoln, and Neeraj Suri. Formally verified on-line diagnosis. *IEEE Transactions Software Engineering*, 23(11):684–721, 1997.