SIXTH FRAMEWORK PROGRAMME
PRIORITY 2
"Information Society Technologies"

Project acronym: DESEREC

Project full title: Dependability and Security by Enhanced Reconfigurability

Proposal/Contract no.: IST-2004-026600-DESEREC

Requirements for translation, deployment and hot reconfiguration modules and for the decision module

Project Document Number: DESEREC/ D31/PU/v1.3 (official version)

Project Document Date: 09/10/2006

Tasks Contributing to the Project Document: T3.1, T3.2, T3.3

Deliverable Type and Security: R-PU

Document Editor : Gwendal Le Grand (ENST)

Author(s): Gwendal Le Grand, Michel Riguidel (ENST), Francisco Hernandez Gomez, Javier Fernandez Sanchez, Pedro Perez Lopez (SGI), Vincent Loriot (EADS), Jean-Etienne Goubard, Alexandre Kirschving, Julien Lemoine, Benoît Bruyère, André Cotton (THC), Manuel Gil Perez (UMU), Véronique Legrand, Jacques Sjaraydaryan, Luc Paffumi (EXAPR), Łukasz Bagrij, (PWR), Marco Aime (Polito)

Abstract: This document presents the requirements for the Translation, Deployment, Reconfiguration, and Decision modules. We present the objectives and model descriptions of each module before focusing on the general constraints for those modules; then, we detail the requirements of each module and of the interfaces with external modules.

Keywords: Requirements, translation module, deployment module, reconfiguration module, decision module, monitoring

History

Version	Date	Description, Author(s), Reviser(s)
1.3	09/10/2006	Version approved by PMT

Executive Summary

The DESEREC project aims to increase the dependability of existing and new networked mission-critical Information Systems (IS) with a Business Services point of view.

The main levers are

- Detect as proactively as possible the incidents and contain them in a limited area of the I.S.
- When an incident causes a failure, handle it smoothly if not seamlessly.
  - ✓ Reconfigure the subpart involved on the same resources or on additional or substituted ones;
  - ✓ When no resource is available to resume performance of a high-priority service in accordance with the business rules, take the risk to stop low-priority services to free the needed resources.
- Use a common model of the Information System to build alternate configurations for forecasted under-nominal modes and validate them through simulation, to analyse the events at various levels, to take the decision to react, and to design the response.

To build the DESEREC framework we propose a three-tiered approach.

- 1) Planning of optimal configuration for anticipated operational modes
  - ✓ Modelling of the networked Information systems, validate resilience and performance of the various configurations with simulation
    - Several minutes to deploy the policies for an alternate configuration
    - Some hours to define a reliable, deployable and optimised configuration and deploy it.
- 2) Fast reconfiguration with priority to critical activities
  - ✓ Reconfiguration, topology
  - ✓ Sustain or resume partial operations in less than a minute
- 3) Incident detection and quick containment
  - ✓ Detection and response
  - ✓ Reconfigure or cut off the area under suspicion within a few seconds

This document is the first step towards the definition and development of the mechanisms for the second tier (Reconfiguration) to ensure the deployment of an operational planning and its hot adaptation following the detection of abnormal events (incident, failure, misbehaviour) on the system.

We separated three functions:

1. *Translation* of an operational planning into actions and parameters to be applied to the subparts.
2. *Deployment* of a defined set of configuration rules and/or to reconfigure some subparts.
3. Upon detection of complex incidents, assistance and guidance to the administrator's *Decision* to drive the reaction process or automatic launch counter-measures.

This document first sketches the objectives and model description of the modules associated to these functions, as the detailed design of the architecture is in process at time of writing.

We move on to express the general requirements on the system; they include language interoperability, scalability, design and operation related to the type of architecture (centralized, distributed or hybrid), security assurance, autonomy of the components, and finally issues related to service continuity and reliability of reconfiguration.

Then, we present the methodology for the expression of the modules' requirements, before describing internal and external requirements (with WP3 and other modules).

Finally, all the requirements are summarized and classified in the last section, based on their priority, and test method.

Contents

Figures

1 Introduction

1.1 Situation of this document

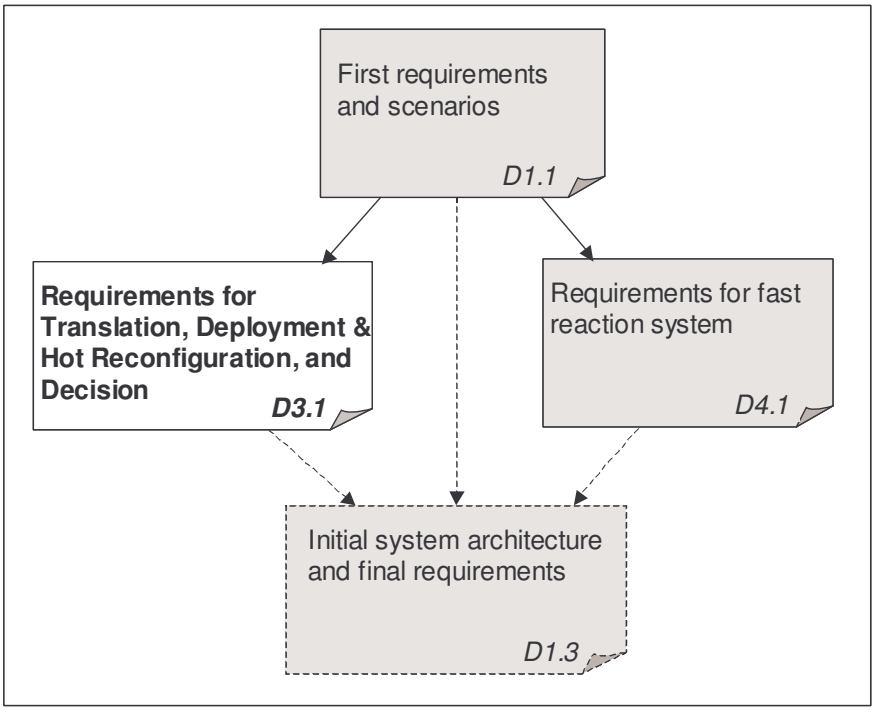

Figure 1. Documentation plan of requirements

Figure 1 presents the situation of this document within the flow of production of the requirements documents down to the Architecture.

1.2 Purpose of this document

The main goal of this document is to put requirements on the Reconfiguration tier of the DESEREC framework. As stated in the technical annex, the main modules are Translation, Deployment, Hot Reconfiguration and Decision.

1.3 Reminder on DESEREC

1.3.1 Objectives

DESEREC will provide Information Systems with increased resilience and robustness by proposing automated deployment, monitoring, and reconfiguration of the services and the systems. This requires to collect data about security and dependability and to design an infrastructure to measure the dependability and security assurance, and to perform decision-making for reconfiguration.

The goal of Work Package 3 is to provide mechanisms to ensure the deployment of an operational planning and its on-the-fly adaptation in response to the detection of abnormal events (incident, failure, misbehaviour) on the system. This work package defines and designs a set of tools for the day-to-day management of complex system allowing:

- Translation of an operational planning to a set of configuration rules.
- Deployment of a defined set of configuration rules or reconfiguration of subparts.

- Detection of complex incidents and means to ensure fast reactions (time scale around the hour) through the use of various detection techniques.
- Assistance and guidance to the administrator to drive the reaction process or automatic launch of counter-measures in order to maintain the system or to better characterise the incident and its impacts.

1.3.2 Functions & modules

The functionalities are provided by three modules, Translation, Deployment and Reconfiguration, and Decision.

The Translator is concerned with the mapping of the high level operational planning that has been set up by the system upon an incident, failure, misbehaviour to an appropriate sequence of concrete management and configuration operations for its various subparts so that the system brings itself to a consistent state.

The Deployment and Reconfiguration module is concerned by the deployment of selected operational modes (after its translation in configuration rules) and their adaptation in response to a detected incident.

The Decision module ensures real-time resilience in operational environment with a reaction time scale of minutes. This quick reaction time will ensure hot reaction in front of identified incident or abnormal behaviour of a critical system. This reaction time includes neither the windows frame to collect events and to detect incidents nor the human process to decide the reaction to apply.

The interaction between the various modules is depicted below (Figure 2):

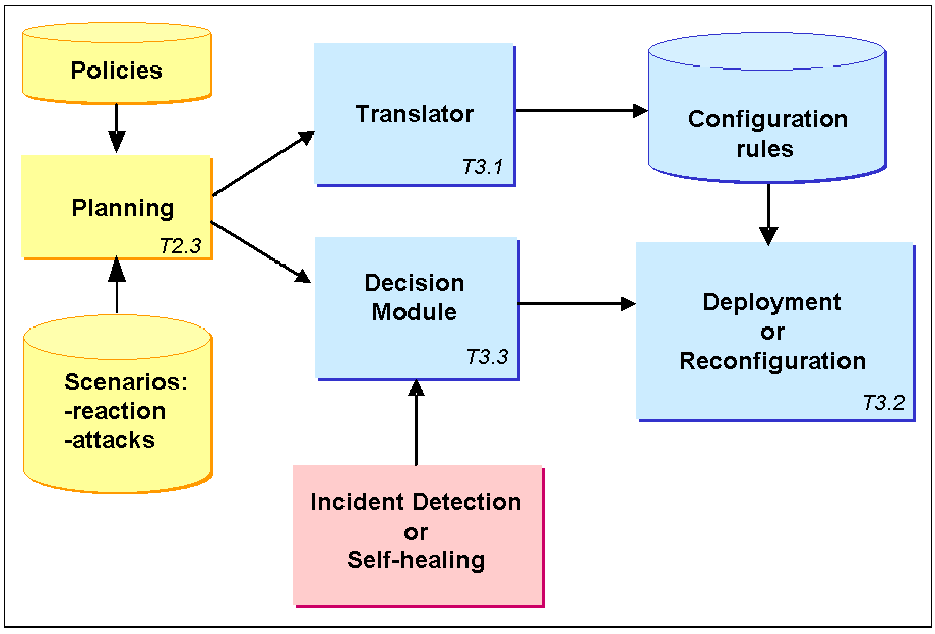

Figure 2. Interactions between modules

In subsequent sections of this document, we describe the methodology and notations used in the document (Section 2), and then we outline the objectives and model descriptions of each module (in Section 3). Next, we present a long term view of the DESEREC architecture (Section 4) before focusing on the general constraints and requirements of DESEREC's modules in Section 5; then, we describe the module requirements in Section 6. Afterwards, Section 7 sums up and classifies the requirements.

1.3.3 DESEREC interfaces

In order to be able to develop a complex set of requirements and methods that will compose the designed interface it is desired to have a general overview on DESEREC architecture:

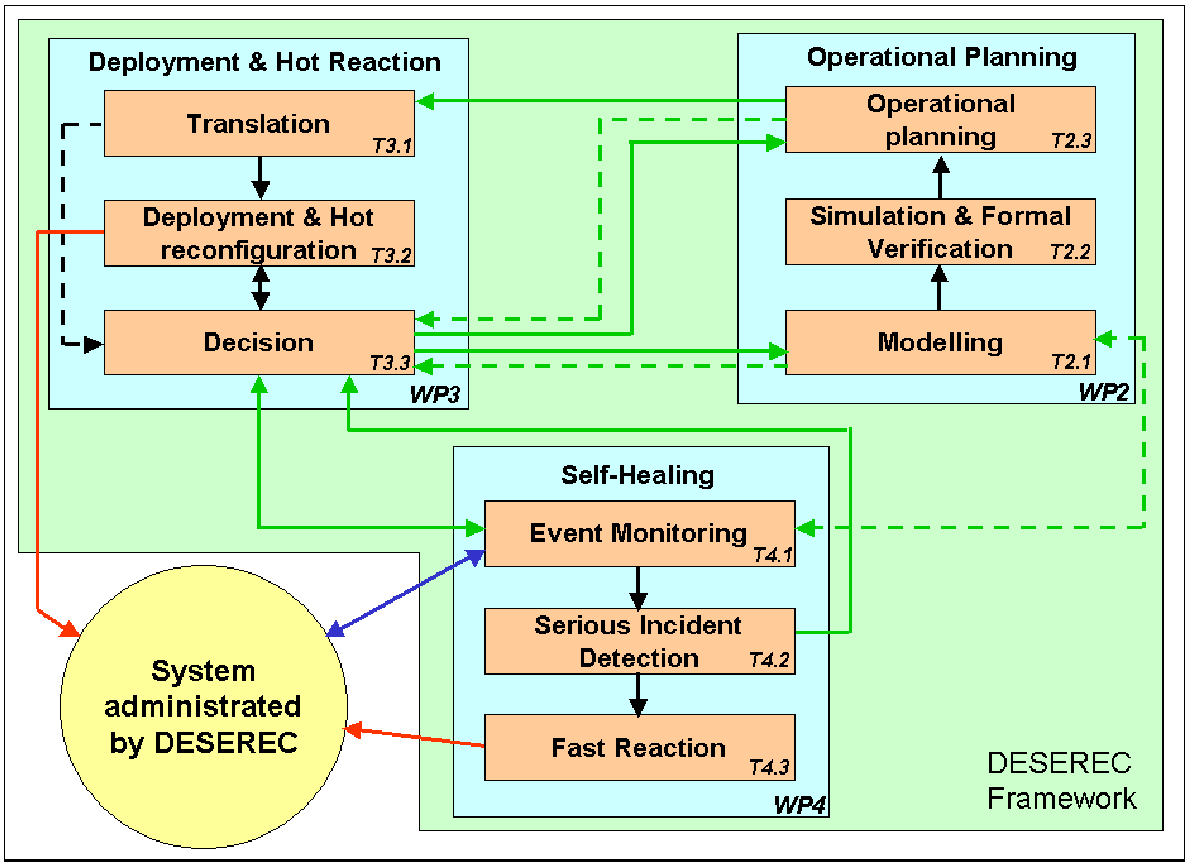

Figure 3. DESEREC first architecture view

Notation:

Internal Work package one/two way interface

External one/two way interface between Work packages

Optional external one/two way interface between Work packages

Optional internal Work package one way

Interface for system reconfiguration tasks

Interface for gathering system information

1.3.4 Work packages

Figure 4 presents the work package breakdown of DESEREC.

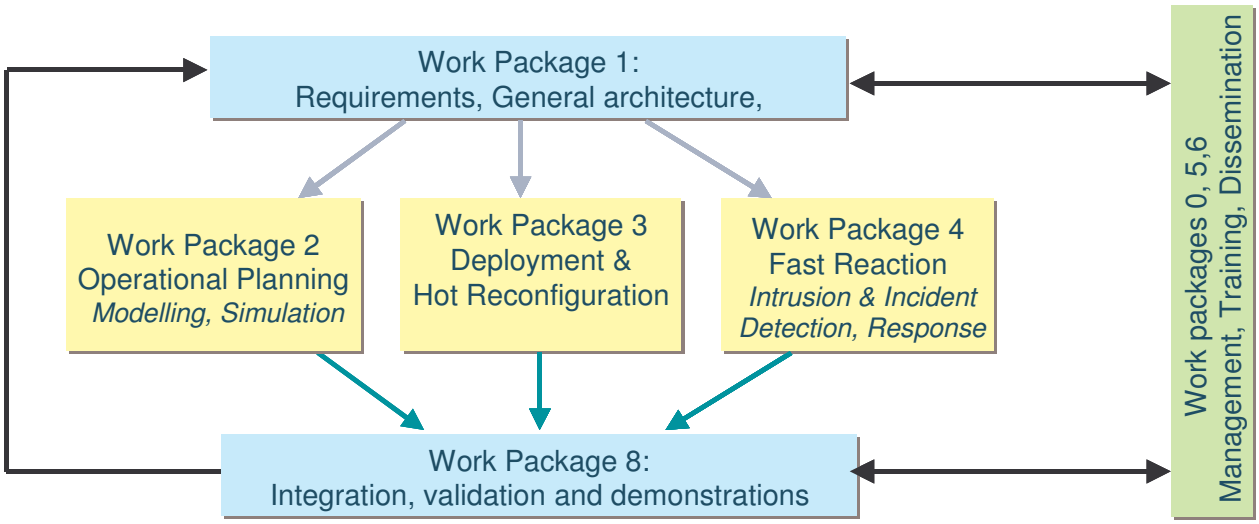

Figure 4. The Work Package breakdown of the DESEREC project

2 Methodology and Notation

2.1 Guidelines for the Identification of Requirements

2.1.1 Definitions

For the overall requirements analysis, WP1 has considered system capabilities and requirements:

- **Capabilities**, which represent a class of common characteristics or features that a system has to provide, for instance security, privacy, etc …
- **Requirements**, which are linked to system capabilities, and provide an additional level of detail.

There are two types of requirements, technical and non-technical:

- **Technical Requirements** describe both the functioning of the system and the operational constraints within which this function is performed and include functional, performance, interface and quality requirements,
- **Non-Technical Requirements** cover other aspects such as contractual understandings, conditions and/or clauses, rules, etc ….

2.1.2 Characteristics of Requirements

A requirement should describe the desired characteristics of the specified system functionality. It illustrates a characteristic of the problem and not a solution. Solutions will be studied later during the design phase. The main qualities of a requirement are that they are:

- **Simple**: A requirement must have an elementary structure and state a basic need. It cannot, at a given level, be broken down into several simpler requirements. At the system level, a simple requirement can be divided into several requirements at Component level.
- **Concise**: A requirement must be expressed in a brief and clear manner. It contains neither an explanation nor a justification.
- **Unambiguous**: A requirement must have only one possible interpretation. A requirement is said to be ambiguous if it can be semantically interpreted in several ways, implying an uncertainty with respect to the design to be elaborated.
- **Verifiable**: As much as possible, it should be possible to verify requirements: if a test can not be associated to each requirement, the validity of that requirement should be further investigated, requirements should be testable either by Inspection or Demonstration or Analysis or Test points. Requirements can be refined in various (testable) sub-requirements to be testable; an effective finite procedure must make it possible to check that the system complies with the requirement.
- **Feasible**: It should be considered if a realistic and satisfactory technical solution can be implemented or can be elaborated in the timeframe of the project.
- **Non-redundant**: A requirement must have no overlap with another requirement. In case of redundancy, requirements must be divided and refined until suppression of the overlap.
- **Non-incompatible**: No conflict with another requirement (see above).
- **Classifiable**: It can be associated with an attribute belonging to a previously adopted classification system.
- **Necessary**: A requirement reflects a need.
- **Traceable**: Identifiable by a unique identifier.

2.1.3 Identification of a Requirement

Each requirement is identified in the following way: "REQ_**Capability_Requirement_n°**" where **Capability** is one word dedicated to express the corresponding capability, **requirement** is one word dedicated to express the corresponding requirement, and **the number** is an incremental identification of each requirement which provide an additional level of detail for traceability purpose. Some explanations, description, example, and schema are added to help understanding of the capability or the requirement and support the definition.

The naming convention is:

REQ_<capability>_<requirement>_<incremental n°>

2.1.4 Levels of Priority

Three levels of priority are defined for the requirements: mandatory, recommended or optional. These levels are indicated by the key words SHALL, SHOULD and MAY respectively. The meaning of the key words is taken from the RFC 2119 and quoted below:

- **SHALL** means "that the definition is an absolute requirement of the specification".
- **SHOULD** means "that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course."
- **MAY** means "that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item."

2.2 Classification of the Requirements

Based on the definitions given, the requirements may be grouped in a table to indicate the relevant information:

- **Priority**: level of priority
  - ✓ M stands for Mandatory,
  - ✓ O stands for Optional,
  - ✓ R stands for Recommended.
- **Test Method**: anticipated test method for the validation of the requirement:
  - ✓ I stands for Inspection,
  - ✓ A stands for Analysis,
  - ✓ D stands for Demonstration,
  - ✓ T stands for Test Point.

Identification	Priority	Test method
XXX_1	R	Inspection
XXX_2	M	Inspection
XXX_3	R	Inspection
XXX_4	M	Inspection
XXX_5	M	Inspection
XXX_6	M	Analysis

Figure 5. Example requirement classification table

3 Objectives and model description

3.1 Translation

3.1.1 Goal of the translator module

The purpose of the translator module is to transform a high level operational plan[1] of a target system resulting from modelling and validating its desired security and dependability properties to concrete sequences of configuration actions for the constituent devices, components, and subsystems, as shown in Figure 6. Moreover, the translator builds the configuration scripts for the components whenever a system reconfiguration is deemed necessary after abnormal behaviour is sensed and according to the policies that are to be enforced to the target system. Requirements on the input formalisms of the translator used for expressing the architecture of the target system and the policies to be applied to it are presented, along with requirements for interacting with other modules of the DESEREC architecture.

Depending on the target system and the platform it runs on, the operational plan may be mapped to different concrete actions for accomplishing the desired configuration. Configuration actions that are expressed as an operational plan have already been simulated and their impact on the operation of target system as well as its dependability properties has been validated and assessed.

The overall steps towards the design of the translator module are:

- To define requirements for translating a high level operational plan to a concrete set of configuration rules. These requirements are presented in this document.
- To specify a formalism for expressing:
  - ✓ Management and configuration capabilities of systems components, as well as constraints and policies for them
  - ✓ Concrete configuration and management plans/rules to be carried out by the system's components
  - ✓ Policies to govern the translation process
- To specify mechanisms for translating an operational plan to a concrete configuration plan
- To design appropriate translation mechanisms

The desired input formalism should constitute a well-structured, highly portable, description language, allowing easy and efficient transformations.

[1] An operational plan is a set of data describing how to configure the system.

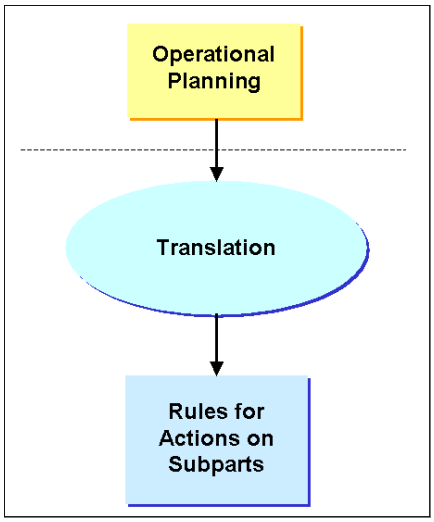

Figure 6: Translating high level operational plans.

The mapping process is driven by a set of rules that specify how high level configuration constructs are to be mapped to low level configuration actions. Possible constraints on the values of configuration parameters, or permissible combinations of them, need also be resolved by the translator. Such constraints should be expressible in an approriate language.

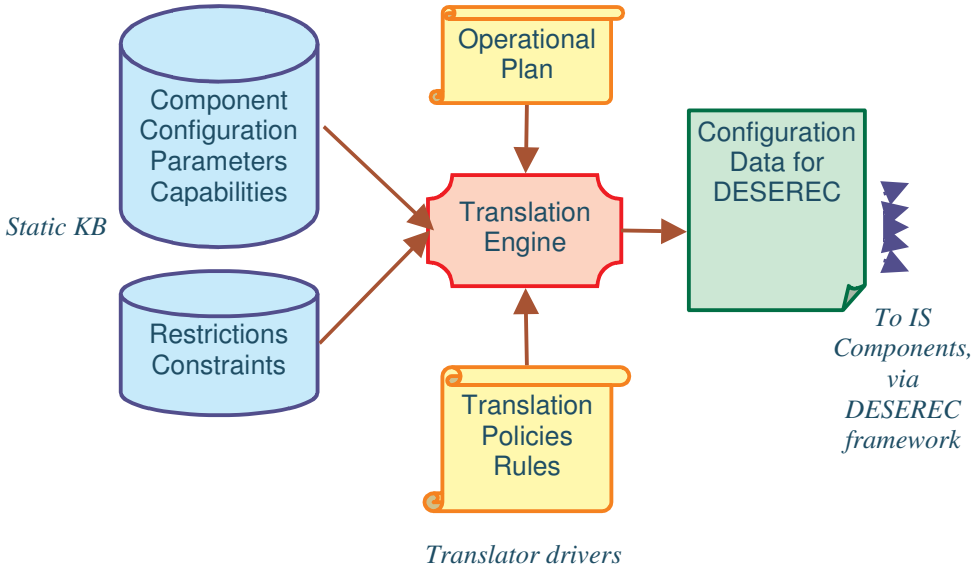

Figure 7: Model of the translation process.

Figure 7 shows a model of the translation process. The translator may use a knowledge base in which are stored the component configuration parameters as well as constraints on their values (if this information is not present within operational plan). Its main objective is to map the operational plan of a target system that results from the modelling, simulation, and verification of its dependability properties to concrete configuration data that are destined to the subparts of the actual system. The outcome of these data is to provide the Deployment tools with enough information to configure the system and/or bring it to the selected operational state.

3.1.2 Interactions with other system modules

The Translation module receives data from the Operational Planning module, performs the translation of these data and makes those transformed data available to the Deployment/Hot Reconfiguration module and other DESEREC framework modules that might require it.

3.2 Deployment and Reconfiguration

3.2.1 Goal and scope of the deployment and reconfiguration module

The aim of the **deployment** is to distribute a new configuration to the target components according to several parameters including the specific configuration and various policies. Deployment may also be used to switch to a degraded mode if it is required by the system. In that case, the latter shall complete a set of critical operations before terminating the switching.

Within DESEREC, we design new concepts for **reconfiguration** automation in order to accelerate the administrator's reaction so as to mitigate the impact of the events and ensure the security of the reconfiguration process (check the integrity of the reconfiguration, check that the reconfiguration cycle has correctly ended, etc.)..

Configuration and reconfiguration can be applied within all the system components, so configuration types may be classified according to the targeted element (the application, the services, the session, network elements, the user, the hardware, etc.) and their typology (management policies, security policies, configurations…).

Within DESEREC, deployment and configuration must therefore be understood in a broad wide sense. They are applied to ontologies: the components are deployed and modified during their life cycle; DESEREC will contribute to the maintenance of the system's components (installation, operation, modifications, etc.). Many classifications for the reconfiguration of ontologies have been proposed in the literature. For instance, Buckley et al.2 evaluates the ability to specify the possible configurations according to a two-dimensional grid: reconfiguration operations as one dimension and architectural elements available to be used in the change as the other. It presents its own taxonomy with a concept artefact, granularity and impact. Artefacts represent source code, file, executable code, etc. or components, connectors. There are many possible artefact instantiations that can be divided into three types3:

- *Basic Reconfiguration Operations*. The basic reconfiguration operations are component addition, component removal, connector addition, and connector removal. For example, the use of scripts is one means to achieve composite operations.
- *Composite reconfiguration operations*. It represents the basic operations in a composite form is also considered. For example, the use of scripts is one means to achieve composite operations. In composite operations we consider not only the ability to add or remove subsystems or groups of architectural elements but also the constructs that can be used in specifying the operation.
- *Variability of Architectural Elements*. The set of architectural element instances (the components and connectors) involved in the change can be fixed to the set of elements included with the software system prior to run-time, or the set can be variable in components and connectors added during run-time.

3.2.2 Technological issues

Possible candidate technologies to execute those services include:

- **Policy handling methods**. Policies offer the possibility to configure a system depending with the rules based installation controlled by a rules file. The rules file contains policies for each group of systems that should be automatically installed.

---

[2] Jim Buckley, Tom Mens, Matthias Zenger, Awais Rashid, and Günter Kniesel. Towards a taxonomy of software change. Journal of Software Maintenance and Evolution: Research and Practice, 2004.

[3] Jeremy S. Bradbury - "Organizing Definitions and Formalisms for Dynamic Software Architectures" - Technical Report 2004-477

Security policies compliance is the solution able to handle validation for new changes against appropriate security policies and rollback to previous configurations for non-compliant changes.

- **Configuration management by agent**: an agent is deployed on each component and processes are managed by agents. This solution proposes preparation and installation. Moreover, it is able to locally collect information and execute commands. It provides scalability and flexibility of configuration management but requires a preparation and expert intervention.

One of the major issues is therefore to design **self-configuring** mechanisms and improve **self-healing** properties of the systems. For many years, companies have deployed their systems with traditional solution requiring human operations. This solution provides integrity, scalability and flexibility of configuration management but requires a very heavy preparation and expert intervention. DESEREC aims at facilitating, assisting and automating this procedure as much as possible. This concept dynamically implies that the IT infrastructure the ability to configure or reconfigure itself on the fly, which is a natural continuum with respect to current trends since the last decade was the renaissance in the area of the self-repairing computing concept.

Two other major issues that the Translator should address are: collecting (re)configuration feedbacks, and track reconfiguration history.

3.2.3 Interactions with other modules

The interaction between the *Decision*, the *Deployment* and *Reconfiguration* modules is depicted hereafter:

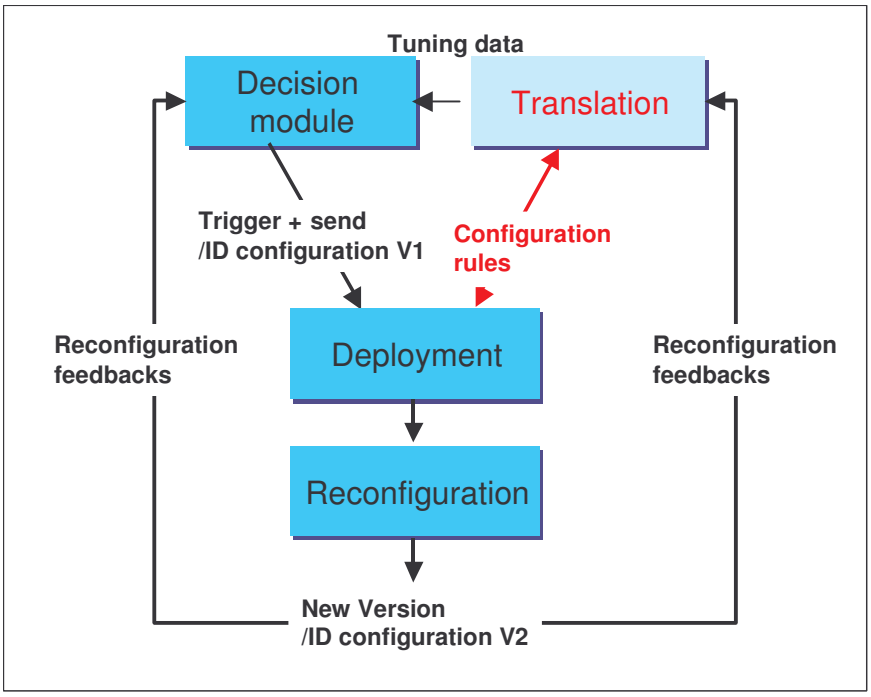

Figure 5. Interaction between the decision, deployment and reconfiguration modules

The *Deployment* module consists in packaging a configuration rule into a deployable file or executable commands. It receives a command from the decision module (e.g. push, put or update a given configuration or a given command to one or many components or devices). This order is used to update a specific system or service. Moreover, the update process must be executed by hot reconfiguration of the various components; in other words, it must preserve all the operational services and reconfigure unavailable or degraded services.

Obviously, the update should be done reliably in the sense that there are means to verify that a user, application, service, network or hardware update is not corrupted and can be or has been applied correctly. Once a modification has been applied to the system, a final process should inform other modules about the configuration action status (e.g. success, failure, error with specific feedback). DESEREC's objective in this context is to design tools to assist the deployment and reconfiguration of the system, to maximize the reliability of the tools, and facilitate their monitoring.

Since DESEREC will manage the Information System's components through a global vision, this raises many questions related to reconfigurability:

- What is the best environment and architecture so that components collaborate and communicate with high-level management tools?
- To what extent should components be autonomous (regulate themselves)?

3.3 Decision

This module has three main objectives:

- Monitoring the whole system and its services in order to evaluate the current state and the value of dependability attributes,
- Performing in depth analysis of system events to detect known and unknown situations. A self-learning process will help to define and refine new reaction scenarios.
- Selecting the reaction scenario to be applied for an optimal reconfiguration of the system and its services. The decision module will assist the decision process by proposing several reactions scenarios to the system operator (who will make the final decision).

The first two points are referred hereafter as the detection functions and the last one includes both the reaction selection function and the user interface.

3.3.1 Simple vs. complex events detection

Current systems provide local and specific solutions for monitoring, alerting and management of security assets. Within DESEREC, we will design a system that consolidates different events (network traffic, user management, application events, etc.) and builds a whole dependability picture, thus providing useful decision support information. Thus the detection of incidents is performed at two levels:

- Simple events detection by *Detection* module: Use simple rules to detect the occurrence of basic incidents by matching log information with simple patterns. Incidents reported at this level are spawned by simple events or by simple aggregation / correlation of basic events. Therefore, events involved in correlation process are system events (non business events), contextual information associated to events is not taken into account for correlation process, and, finally, this correlation is based on simple patterns and easy-to-design rules.
- Complex event detection by *Decision* module. The aim in the Decision module is to make easy taking decisions by detecting complex incident and showing the appropriate reactions to the operator. To be successful, it is critical to filter basic incidents and promote only the relevant ones, avoiding the operator to be flooded with so many events. The key is to obtain important incidents taking into account the relevance of the system / process affected by the incident. Therefore, several types of events can be also involved in correlation process, contextual information associated to events must be taken into account for correlation process, and finally, this correlation is based on complex rules.

3.3.2 Complex events detection process

Therefore, the *Decision* module should at least:

- Correlate low-level events to identify system operational states
- Correlate events and alarms to identify global faults (match threats)
- Select new protection profiles (large faults) and new system configurations (minor faults), trigger proactive periodic changes, and select specific reaction steps (sent to the *Deployment* module)
- Log incident (and security related events) history
- Log reconfiguration history (with feedbacks)

The event format based on CBE / IDMEF definitions is a candidate format for both, simple and complex events so that complex events obtained by correlation can also be stored into the same database as the one used for simple events. Thus, the engine could take the events and incidents from the database and correlate them in order to obtain new incidents by applying its rules.

In order to better describe these high level objectives, three main features of the module have been identified: detection process, reaction selection and user interface. In addition, some interface requirements complete the scope of what the *Decision* module should do.

3.3.3 Reaction selection

The results of the detection, i.e. the proposed matching reactions, will be analyzed by the operator to select an appropriate reaction.

In nominal cases, by using the knowledge associated with each reaction scenario generated by the *Operational Planning* module, the detection module should propose to the operator a small set of candidate reactions to the identified incident, and then the operator should choose in the list which reaction is the most appropriate (this process should be as automatic as possible and may not require operator intervention).

Sometimes, the operator does not find what he wants in the selected list, or the incident is an unknown one (not mapped to a reaction); so, we must address what the system should do in such cases.

The last functions of this capability expressed in this document are related to the storage of the reaction selection attributes: who does it? When? How?

3.3.4 User interface

Regarding the previous capabilities, an operator will act in the following manner:

- Detection tuning which prepares the detection or improves it depending on the technologies used.
- Reaction selection which consists in the human aided choice of the appropriate reaction scenario regarding the detected incident.

The operator should be assisted by a monitoring view in order to:

- Observe the system and services operational states with zooming capabilities. In fact, the interface should be rich so that the operator may manage large amounts of data.
- Decide an appropriate reaction scenario with respect to the detected incident.

In DESEREC, the best quality of the critical services is expected; this can be done by gathering different kinds of information and by providing the best relevant and consistent views to the operator (administrator of the DESEREC system). So in addition to the dedicated user interface for detection and decision purpose, a general and centralized ‘monitoring console’ will be designed into this module of DESEREC. This console is responsible for monitoring changes in the system by interacting with sensors deployed in the infrastructure. It provides:

- An overview of the operational states of the system.
- A diagnosis view to analyse the alarms and incidents that occur.
- A trust assessment for decision making before reallocating resources to mitigate threats.

In order to achieve these objectives, the different kinds of information needed are:

- Regarding the components QoS and states views :
  - ✓ The low level measures (events, alarms, …) of the traffic, activity, load, …
  - ✓ The high level measures of the security, dependability and trust in components.
  - ✓ The infrastructure view of the system
- Regarding the services QoS and states views :
  - ✓ The high level measures of the security, dependability and trust in services.
  - ✓ The service composition and possibly their related workflows (this is required for detection scenarios and threat matching)

4 "Beyond the horizon" trends in Information Systems resilience

Reconfiguration concerns not only the states of the network elements, but also other layers in the system. The main goals for the design of resilient systems are to foresee the development flaws, detect trends or anomalous behaviours to proactively manage the system in order to prevent serious problems from arising, install prevention measures, and reactively control the system by making fast adjustments in response to the changes within the system or its environment to stop fault proliferation. Therefore, it is important to model the system itself as well as the security management. The basic needs are motivated by security functional requirements (prediction, scenario simulation, prevention, monitoring, autonomy, reconfiguration, etc.).

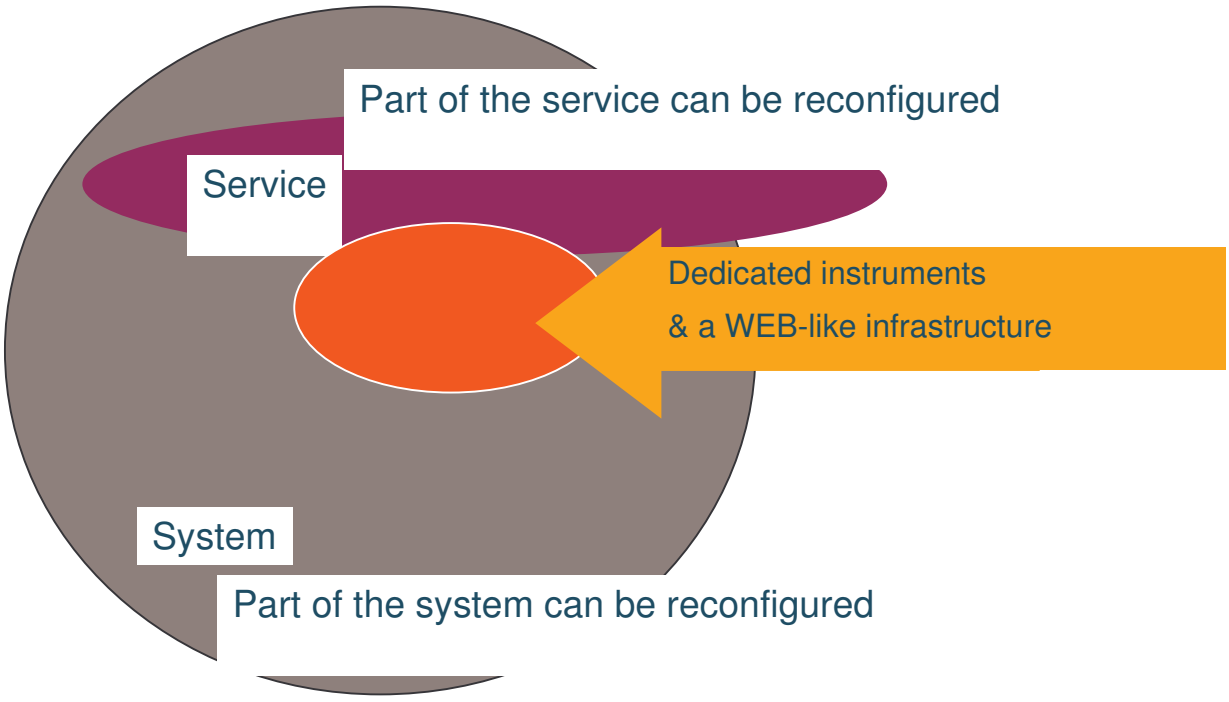

Figure 8. Overall view of Reconfiguration

It is necessary to model the system and define what state it is currently in and evolving to.

Two types of states are identified: predefined states (for known events) and undefined states (when unknown events are detected) according to a set of rules. Since the system must be configured and reconfigured on the fly, it is mandatory to identify what configuration or reconfiguration should be applied, and then how to enforce it. Therefore, modelling of security, dependability, resilience, and assurance of the complex ICT system will play a crucial role. The different components and features of ICT systems and supported services shall be modelled abstractly to be considered within the rules that manage the system and its hot reconfiguration.

However this is not sufficient to ensure system and service survivability. Abstraction is the process of simplifying irrelevant details according to the requirements and focusing and generalizing what remains. It makes descriptions more concise and understandable, helps in concentrating on the important and assists scalability of resulting models. To obtain a relevant model, our consideration of the system should not be limited to it components and to the offered service types. The dynamicity of complex ICT systems (with their states, modes, domains …) makes them fragile, and vulnerable. Likewise, the fast introduction of abstract services (with phases along time) into the market makes them fragile and vulnerable. Therefore, to ensure

system and service survivability, security, dependability and resilience assurance shall be taken into account and then modelled together with systems and services. Many vulnerability and assessment methods have been studied in the literature, but there is little use of these methodologies by day-to-day IT operations staff. Products often include software tools that address specific IT platforms, and lack the overall security assessment ability. Practical and business-related methodologies that can bridge this gap are required

More formally, we can thus express the following modelling general constraints for self-configuration/reconfiguration of ICT systems:

- Abstract and dynamic ontologies shall be modelled to represent the system (topologically and virtually, including the subsystems) architecture, the system composition (the hardware, software and data composition), the network architecture and components, the services (business, policy, contract rules, flows, sessions), the configuration architecture (configuration entities, operations)
- The measurable properties of the dynamic systems with virtual constructions shall be identified to estimate the level of the security, dependability and resilience assurance of the system using metrics.
- QoS, security, mobility interoperability and reconfigurability constraints and protocols shall be modelled

However, the question of assurance modelling for the (complex ICT) systems and their abstract services remains a problem. In flexible dynamic systems with virtual constructions, the properties to measure are not clearly identified. Thus, it is essential to identify them and to define the various metrics useful to determine the level of security, dependability and resilience assurance of the system.

Hot reconfiguration of the system when errors are detected involves a “real time” estimation of the (security, dependability and resilience) assurance level of the system. Therefore, **it is essential to collect, to measure and to estimate the direct actual parameters of the established models in “real time”**. Once, the measurements are collected and estimated, several problems are raised. For instance:

- What is the confidentiality of the data? To whom should they be revealed and when?
- The computation of the measured metrics to represent the assurance of a composition of a graph of several entities (server + session + flow) with several subjects (players, actors, customers …). This task is complex due to the number of parameters at stake in the various possible scenarios.
- **Thresholds** in the various measured variables must be defined to determine how and when to decide and react to an error state of the system to then ensure the survivability of the system and of the offered services.

All the constraints shall be incarnated and instantiated within the network architecture, the protocol specification, the applications, and markers in case of parameters or programs for example. These constraints are **distributed** within the whole system.

This may only be achieved by adopting a general vision of the system in which the entities are not hardware/software or computer/network elements but abstract service related components. This raises many issues related to:

- Interoperability of language and networked components,
- Hybrid obligation: the models and reconfiguration in the system should be cross-plane, as well as partly autonomous and distributed
- Granularity, view, level of abstraction, scalability
- Relevance, level of importance,
- Threats and security assurance,
- Time scale,
- Etc.

5 General considerations

Several aspects shall be considered in the description WP3 modules. They are organised in general constraints and general requirements.

5.1 General constraints

5.1.1 Interoperability

Interoperability of the system's components is a critical issue in order to achieve a global view. A common vehicle (language) will be necessary for components to exchange consistent information between heterogeneous managed components. Actually, multiple existing tools are able to apply reconfiguration on networks, systems, databases or applications, but only few are able to apply any configuration to any component. This language should allow managing all kinds of configuration as similarly, a variety of configurations exist on a component: application, network, system, user, etc.

Interoperability of the components is a prerequisite to enable the deployment and reconfiguration features that are envisaged, since information must be exchanged among heterogeneous components and/or architectures (management, system, applications …).

It is necessary to use as much as possible standard protocols which can help in the reconfiguration process during the exchange of information between components and the management framework. Those protocols should allow management applications to define several functions: to read management information from components in a standard way, to apply required management information to components, to run remote methods or to trap events related with managed services.

5.1.2 Scalability

Our ability to model, understand and deploy large-scale networks and systems is generally limited by the incomplete understanding of the fundamental driving forces that affect their evolution. Actually, the number of possible interactions between components increases dramatically as the number of components grows. As a result, the activity of large networked systems greatly exceeds the ability of a single centralised entity to evaluate, monitor or manage them in real-time. The most promising way to address is the issue of scalability is by designing distributed solutions since our large system will produce a huge amount of events and logs that must be processed.

The infrastructure models should at least represent the topology and behaviours. They will cope with a set of several data and parameters that will be selected according to current needs. Then, an adapted abstraction process will consist in selecting relevant parameters for the description of the system, which will assist the scalability of the model. In a hierarchical description of the infrastructure, it is likely that it will be necessary to consider different sets of relevant parameters for each level.

5.2 General requirements

5.2.1 Threats and security assurance

Real failures occurrences are not equivalent regarding threats perception: on the one hand, accidental physical faults and malicious attacks occurs less seldom than predicted, on the other hand, system failures due to software upgrades, organizational faults (distrusted system administrator) and service contract faults (SLA – Service Level Agreements – not handling crisis situations) are not often handled by system administrators but raise the majority of system failures.

These considerations will have a significant impact on the input events used for the detection process.

5.2.2 Time scale

The detection and reconfiguration modules should be part of a detection/reaction process that takes a maximum of a few hours.

This can be done if the system has a limited amount of data to gather (to reduce processing time) and if the time interval used for detection is small (because we may need to correlate the events during a long period).

5.2.3 Autonomy and Control

A possible approach to deployment is the use of intelligent autonomous agents. Those smart components should observe the environment, communicate, report to third parties, and perform actions to reconfigure the system and modify the allocation of resources. Obviously, even if the system is hierarchical, reconfiguration concerns all the levels within the system, and thus these dynamic components must be located at all the levels of abstraction, express dependencies from different models, and can co-operate with each other.

Each would also adapt to the environment as it is modified by external components and evolutionary changes. Such adaptation may be defined as the capacity for modification of goal-oriented individual or collective behaviour in response to changes in the environment. This modification may be either an experience-based response (selecting an optimal pre-determined pattern) or a new response (developing new responses not previously known to that agent), depending on the feedback reported by these intelligent and dynamic components. Moreover, those components should have the ability to proactively and autonomously deny a new service request or to reduce some quality of service metric in the context of some Service Level Agreements in order to ensure the future viability of the system. No single entity has complete control of multi-scale, distributed, highly interactive networks, or the ability to evaluate, monitor, and manage them in real-time. Management of disturbances in infrastructures requires basic understanding of true system dynamics, as well as distributed control. After a disturbance, parts of the networks have to remain operational and automatically reconfigure themselves.

5.2.4 Service continuity and security of reconfiguration

DESEREC will provide hot-reconfiguration and deployment features. In other words, a dynamic reconfiguration of the system's components that does not require service interruption shall be available. Existing hardware-based solutions allow for the dynamic reconfiguration of applications, by replacing the existing system with an updated one running on a secondary machine. These systems come from the dependability and fault tolerance fields. Within this project, the new challenging issues will concern the grain of reconfiguration (and the self-reconfiguration features) since the latter may concern any number of components from a single element to the whole system.

Moreover, service continuity after reconfiguration also raises the issue of reconfiguration security: DESEREC will need to propose mechanisms to guarantee the integrity of the requested reconfiguration since it could provoke a breakdown of the service if it is corrupted. In addition, several reconfigurations should not be applied simultaneously and the administrator should be able to verify that the (re)configuration cycle has correctly finished.

6 Modules Requirements

Based on the constraints mentioned above and the objectives of the DESEREC tools, we have extracted the following requirements for WP3 modules.

6.1 Translation module specific requirements

- ***Components configurations and classification***. It should take into account the following types of configurations
  - Static or direct: For example, if the same path is always used when accessing configuration files.
  - Dynamic or indirect: For example, if real configuration values are stored in environmental variables.
  - Manual: If an operator is required to perform the configuration.
  - Automatic: If configuration can be performed in an unattended mode.
  - Global: The same configuration must be applied to all components of the same type.
  - Individual: The configuration must be applied to a concrete component (regardless of the rest).
  - Dependent: The configuration applied to one component has an impact on other components.
  - Hot configurations: it's necessary to take into determine if the component must be restarted or not after a configuration has been applied.

When configuring a component or a whole system, (inter-)dependencies among values of configuration parameters may result into conflicts that should be resolved. Constraints on parameter values should be expressed and represented formally, having the translator resolve them during the translation phase.

Requirement identification: **Translation_Component_Groups**

- **REQ_Translation_Component_Groups_01**: The translator SHALL allow to associate into groups a set of components with similar characteristics.
- **REQ_Translation_Component_Groups_02**: The translator SHALL be able to produce configurations that are not dependent of the component localization.

Requirement identification: **Translation_Configurations**

- **REQ_Translation_Configurations_01**: The translation module SHALL output configuration rules that can be applied to specific components.
- **REQ_Translation_Configurations_02**: The translation module SHALL specify configuration both for individual components and for component groups.
- **REQ_Translation_Configurations_03**: The translation module SHALL manage dependencies related to configuration changes.

Requirement identification: **Translation_Constraints**

- **REQ_Translation_Constraints_01**: The translator SHALL allow the formulation of constraints that are imposed on specific components (restrictions on the allowable primitive actions and on the allowable values of configuration parameters, as well as on combinations of the above).

Requirement identification: **Translation_Output**

- **REQ_Translation_Output_01**: The translator SHALL build a configuration plan as a set of actions destined for the components of the target system.

6.2 Deployment specific requirements

Requirement identification: **Deployment_Configuration**

- **REQ_Deployment_Configuration_01**: The module SHALL deploy configuration provided by the translation module.
- **REQ_Deployment_Configuration_02**: It SHOULD ensure that the configuration is correctly applied.
- **REQ_Deployment_Configuration_03**: If configuration can't be correctly applied, it SHOULD ensure that the component is in the state prior to the reconfiguration.
- **REQ_Deployment_Configuration_04**: It SHOULD provide a rollback mechanism in order to restore the component to an older valid configuration version.

Requirement identification: **Deployment_Performance**

- **REQ_Deployment_Performance_01**: The operational plan SHALL be deployed in less than an hour.
- **REQ_Deployment_Performance_02**: It MAY use robust deployment mechanisms in order to be tolerant to a predefined number of DESEREC components failures.

Requirement identification: **Deployment_Monitoring**

- **REQ_Deployment_Monitoring_01**: It SHALL report results concerning modifications in the system (success or failure).
- **REQ_Deployment_Monitoring_02**: It SHOULD provide information regarding deployment progress at the system level.

Requirement identification: **Deployment_Security**

- **REQ_Deployment_Security_01**: The system SHALL provide security mechanisms (ciphering, authentication …) to deploy configuration.
- **REQ_Deployment_Security_02:** A security mechanism SHALL be implemented to protect the access to T32 data from unauthorized access.

Requirement identification: **Deployment_Audit**

- **REQ_Deployment_Audit_01**: Each deployment SHALL generate audit tracks regarding at least the component and the deployed operational plan.

6.3 Reconfiguration specific requirements

The configuration process updates bad or inexistent configurations towards a good configuration. Its role is limited to applying reconfigurations. The configuration process is split into two major processes:

- Deployment process: storage, selection and transfer suitable configuration towards given component,
- Feedback process: all functions are able to assure integrity of last known good configuration

In DESEREC, the fast cicatrisation applies configurations in fast mode but with its local view. As reconfiguration includes all actions only triggered by the Decision module, in this case, the decision module has a global view.

The deployment and reconfiguration phase is the last step of the entire process. This is a critical phase which needs much control. An entire process called feedback process is dedicated to this control.

Requirement identification: **Reconfiguration_ServiceContinuity**

- **REQ_Reconfiguration_ServiceContinuity_01:** There SHALL be mechanisms to ensure that reconfiguration can be applied without interrupting running critical services.

Requirement identification: **Reconfiguration_Storage**

- **REQ_Reconfiguration_Storage_01:** There SHALL be a mechanism that keeps the up-to-date configurations.
- **REQ_Reconfiguration_Storage_02:** There SHALL be a mechanism that keeps the configuration plans received from the Decision module.
- **REQ_Reconfiguration_Storage_03:** There SHALL be a mechanism that keeps the up-to-date feedbacks that SHALL be sent to the Decision module in order to track feedbacks.
- **REQ_Reconfiguration_Storage_04:** The integrity of databases SHOULD be ensured.

Requirement identification: **Reconfiguration_Supervision**

- **REQ_Reconfiguration_Supervision_01:** There SHALL be a mechanism that supervises the system in order to report failures directly to an operator. Note: this mechanism may be shared with WP4.

Requirement identification: **Reconfiguration_Communication**

- **REQ_Reconfiguration_Communication_01:** There SHALL be a communication protocol to communicate between components.
- **REQ_Reconfiguration_Communication_02:** Communication between components SHOULD be reliable.

Requirement identification: **Reconfiguration_Handle**

- **REQ_Reconfiguration_Handle_01:** There SHALL be a mechanism for managing versioning configurations.
- **REQ_Reconfiguration_Handle_02:** There SHALL be mechanisms that verify the completion of prerequisites before installing new configurations.
- **REQ_Reconfiguration_Handle_03:** There SHALL be mechanisms that verify if the configuration was correctly applied.
- **REQ_Reconfiguration_Handle_04:** There SHOULD be mechanisms to rollback to previous good configuration versions in case of a local failure.
- **REQ_Reconfiguration_Handle_05:** There SHOULD be different levels of packaging modes allowing to send batch reconfiguration actions to a unique component
- **REQ_Reconfiguration_Handle_06:** There SHALL be mechanisms that manage time (e.g.: time allowed for each task).
- **REQ_Reconfiguration_Handle_07:** There SHOULD be mechanisms that monitor and assess processes at any time.

Requirement identification: **Reconfiguration_Feedback**

- **REQ_Reconfiguration_Feedback_01:** There SHALL be mechanisms to report both success and failures of configuration installations.
- **REQ_Reconfiguration_Feedback_02:** There SHOULD be mechanisms that manage corrupted (or genuine) feedbacks from components.
- **REQ_Reconfiguration_Feedback_03:** There SHALL be mechanisms that identify the outcomes of the reconfiguration.
- **REQ_Reconfiguration_Feedback_04:** There SHALL be mechanisms that construct the responses (in a unique data format) sent to the Decision module.

Requirement identification: **Reconfiguration_Performance**

- **REQ_Reconfiguration_Performance_01**: There SHOULD be mechanisms that provide High-Availability for the deployment phase.

Requirement identification: **Reconfiguration_Security**

Requirements for the security of reconfigurability have already been expressed as REQ_Deployment_Security_XX

Requirement identification: **Reconfigurability_Audit**

- **REQ_Reconfiguration_Audit_01**: Each reconfiguration SHALL generate audit tracks.

6.4 Decision module specific requirements

Decision support tools for infrastructure management will facilitate decisions in the phases of regular operations, malfunctions and other incidents. They will support at least:

- Optimal use of existent systems, subsystems, and units,
- Determination of capabilities necessary in different operations.

Based on the subdivision proposed in the previous section, we express the following requirements:

6.4.1 General requirements

This subsection contains functional and non-functional requirements about Decision Module processes in general.

Requirement identification: **DecisionModule_Storage**

- **REQ_DecisionModule_Storage_01:** The Module SHALL maintain events and alarms that occur.
- **REQ_DecisionModule_Storage_02:** The Module SHALL maintain detection, decision and reconfiguration data associated with incidents.
- **REQ_DecisionModule_Storage_03:** All the data stored SHOULD be available during two days.

Requirement identification: **DetectionModule_SystemView**

The decision module should build a high level view of system operational status

- **REQ_DetectionModule_SystemView_01:** The decision module SHOULD build a high level view of critical subsystem operational status.
- **REQ_DetectionModule_SystemView_02:** The decision module SHOULD build a high level view of service operational status.
- **REQ_DetectionModule_SystemView_03:** The decision module SHOULD build a high level view of DESEREC infrastructure status (agents not responding).
- **REQ_DetectionModule_SystemView_04:** The decision module SHOULD build a high level view of internal and external changes to the system configuration.

6.4.2 Detection process

This subsection contains functional and non-functional requirements about detection tuning, detection process (performance, relevance ...), detection results analysis, detection triggers (for synchronizations purpose)...

Capability: **Detection Process (DetectionProc)**

Requirement identification: **DetectionProc_EventManagement**

- **REQ_DetectionProc_EventManagement_01**: There SHALL be mechanisms to allow the system to analyze up to 24 hours of events to detect incidents.
- **REQ_DetectionProc_EventManagement_02**: It SHOULD be possible to configure the length of the window time within which events will be processed.

Requirement identification: **DetectionProc_Pattern**

- **REQ_DetectionProc_Pattern_01**: The detection module SHALL match or recognize patterns in a flow of events.

Requirement identification: **DetectionProc_Investigate**

The operator needs some event-level information to understand the cause of detected incidents.

- **REQ_DetectionProc_Investigate_01**: A mechanism MAY assist the operator to understand the cause of detected incidents.

Requirement identification: **DetectionProc_Knowledge**

When the operator has prior knowledge (for example in the form of rules) of the system, it should be included in the detection process.

- **REQ_DetectionProc_Knowledge_01**: The Decision module SHOULD include an interface with which the operator can change the correlation parameters.
- **REQ_DetectionProc_Knowledge_02**: Mechanisms SHOULD be designed in order to reduce false positives.

Requirement identification: **DetectionProc_Performance**

- **REQ_DetectionProc_Performance_01**: The detection SHALL be finished in a time scale of a few hours after the last events of an incident.
- **REQ_DetectionProc_Performance_02**: High availability of the detection systems SHOULD be ensured

Requirement identification: **DetectionProc_UnknownSequence**

- **REQ_DetectionProc_UnknownSequence_01**: A mechanism SHOULD be implemented in order to identify unknown incident patterns.

Requirement identification: **DetectionProc_Security**

As the detection process contains all the data of the system, these data should be encrypted or stored in a secure database.

- **REQ_DetectionProc_Security_01**: A security mechanism SHALL prevent unauthorized access to events collected from the system.

Requirement identification: **DetectionProc_Handle**

If a part of the IT infrastructure is down, the detection process will have missing values but the detection process should keep on working for the rest of the system.

- REQ_DetectionProc_Handle_01: The detection process MAY be capable of handling missing and noisy data depending on the detection technique used.

Requirement identification **DetectionProc_SystemView**

- **REQ_DetectionProc_SystemView_01:** The decision module SHOULD build a high level view of system operational status including: critical subsystem operational status, service operational status, DESEREC infrastructure status (components not responding), track internal and external changes to the system configuration.

6.4.3 Reaction selection

This subsection details how we will map the detection and reaction scenarios (for known and unknown incidents), and all processes related to the user interaction regarding reaction selection

Capability: **Reaction selection (ReactionSel)**

Requirement identification: **ReactionSel_Mapping**

- **REQ_ReactionSel_Mapping_01**: The module SHALL propose one or several reactions based on the detection and reaction scenarios.
- **REQ_ReactionSel_Mapping_02**: There MAY be mechanisms to deal with situations where no reaction is associated to an incident.
- **REQ_ReactionSel_Mapping_03**: Mechanisms SHALL trigger the deployment of the configuration selected by an operator.
- **REQ_ReactionSel_Mapping_04**: Mechanisms MAY propose different reconfiguration scenarios depending on assurance thresholds depending on the mapping specification available in reaction scenario attributes.
- **REQ_ReactionSel_Mapping_05**: A mechanism SHOULD enable the operator to choose a reaction which is not proposed automatically.

Requirement identification: **ReactionSel_Monitoring**

- **REQ_ReactionSel_Monitoring_01**: It SHOULD be possible to assess all the information related to incidents prior to selecting the reaction to apply.
- **REQ_ReactionSel_Monitoring_02**: It SHALL be possible to know the status and outcomes of the reaction process applied.

Requirement identification: **ReactionSel_Audit**

- **REQ_ReactionSel_Audit_01**: All the reactions applied SHALL generate audit tracks.
- **REQ_ReactionSel_Audit_02**: The system SHALL keep track of who (operator) has selected a reaction scenario

6.4.4 User interface

User Interface(s) is (are) considered as a set of specific functional requirements of the task; these requirements will cover: the way to provide an operational view of the system (services and component views of their states, security and dependability levels ...), and the different control panels dedicated to the operator interactions for detection and reaction processes.

Requirement identification: **UserInterface_Administration**

- **REQ_UserInterface_Administration_01**: A graphical user interface SHOULD provide an easy administration and deployment of reactions, and decision-making

Requirement identification: **UserInterface_Reaction**

- **REQ_UserInterface_Reaction_01**: The potential damage of the incidents detected SHALL be reported
- **REQ_UserInterface_Reaction_02**: The user interface SHALL provide the list of reactions that could be applied to the incident detected.
- **REQ_UserInterface_Reaction_03**: The user interface SHALL provide a classification of incidents.
- **REQ_UserInterface_Reaction_04**: The cost to return to a normal state after its application SHOULD be reported for each proposed reactions.

Requirement identification: **UserInterface_Security**

- **REQ_UserInterface_Security_01**: The user interface SHALL provide an access control based on the identity and roles of users.
- **REQ_UserInterface_Security_02**: The user interface SHALL use secured channels (e.g. with encryption protocols) between the server(s) and the client(s).

Requirement identification: **UserInterface_Monitoring**

- **REQ_UserInterface_Monitoring_01**: The user interface SHOULD provide an escalation matrix of detected incidents.
- **REQ_UserInterface_Monitoring_02**: The user interface SHOULD provide information about related incidents.
- **REQ_UserInterface_Monitoring_03**: The user interface SHOULD show the evolution of reconfiguration deployment.
- **REQ_UserInterface_Monitoring_04**: The user interface SHALL show the result of the reconfiguration process.
- **REQ_UserInterface_Monitoring_05**: The user interface SHALL provide information regarding the current system configuration (nominal, degraded …).
- **REQ_UserInterface_Monitoring_06:** The user interface SHOULD show the status of DESEREC infrastructure
- **REQ_UserInterface_Monitoring_07**: The User Interface SHALL show the status of managed system infrastructure
- **REQ_UserInterface_Monitoring_08**: The user interface SHALL provide information about the scope and impact of an incident along with the infrastructure affected.

Requirement identification: **UserInterface_DESERECInfrastructure**

- **REQ_UserInterface_DESERECInfrastructure_01**: The User Interface SHOULD show the status of DESEREC infrastructure

6.5 Inter-module Interfaces

6.5.1 Translation-deployment/reconfiguration interface

The output of the *Translation* module corresponds to the final configuration which must be deployed on the target component. The Deployment & Reconfiguration module will later enforce this configuration.

These configuration rules must be stored in an intermediate repository, before being used by the deployment protocols to configure the target components. In order to store this information properly, we have defined an internal format representation, which includes final configuration information and additional information indicating how to manage it.

Requirement identification: **T31-T32Storage**

- **REQ_T31-T32_Storage_01**: The configuration rules SHALL be stored in an intermediate repository before deployment.

Requirement identification: **T31-T32_ConfRules**

- **REQ_T31-T32_ConfRules_01**: The configuration information SHALL indicate to what component the configuration should be applied
- **REQ_T31-T32_ConfRules_02:** The configuration information MAY indicate the absolute path used to deploy the configuration file.

Requirement identification: **T31-T32_Management**

- **REQ_T31-T32_Management_01:** There SHALL be mechanisms to manage sets of configuration rules

Requirement identification: **T31-T32_Communication**

- **REQ_T31-T32_Communication_01**: The translator SHALL retrieve the target system's architecture and the formulated policies that are to be enforced.
- **REQ_T31-T32_Communication_02**: The translator SHALL be able to process updates of an operational plan

Requirement identification: **T31-T32_Translation_Security**

- **REQ_T31-T32Translation_Security_01**: A mechanism SHOULD encrypt information exchanges between the deployment and reconfiguration module and the translation module.

6.5.2 Translation-decision interface (T3.1/T3.3)

This section depicts requirement details about the detection/reaction scenarios and reconfiguration actions feedbacks regarding to this external interface.

Capability: **REQ_T31-T33-Interface**

Requirement identification: **REQ_T31-T33-Interface_Security**

- **REQ_T31-T33-Interface_Security_01**: A mechanism SHOULD guarantee the identity of the components involved in this interface.
- **REQ_T31-T33-Interface_Security_02**: A mechanism to ensure the confidentiality and integrity of the information exchanged SHALL exist between Planning and Validation module and Decision module.

Requirement identification: **REQ_T31-T33-Interface_Detection**

- **REQ_T31-T33-Interface_Detection_01**: The detection module SHALL be able to retrieve relevant detection information for the components to be monitored.
- **REQ_T31-T33-Interface_Detection_02**: An exchange protocol SHALL allow the Decision module to retrieve new detection information added by the Planning and Validation module.
- **REQ_T31-T33-Interface_Detection_03**: A set of reaction rules SHALL be associated to a detection scenario to indicate what actions must take place in the target system when an incident happens.

Requirement identification: **REQ_T31-T33-Interface_Reaction**

- **REQ_T31-T33-Interface_Reaction_01**: The detection module SHALL be able to retrieve reaction information from the decision module when known incidents occur.
- **REQ_T31-T33-Interface_Reaction_02**: A set of components of a system SHALL be associated to a reaction scenario to indicate where to apply the actions that must take place when an incident happens.

Requirement identification: **REQ_T31-T33-Interface_Communication**

- **REQ_T31-T33-Interface_Communication_01**: A communication protocol SHALL exist between the translation and the Decision module.
- **REQ_T31-T33-Interface_Communication_02**: A mechanism SHALL publish all necessary data for the self-learning and modelling processes (such as incident history, reconfiguration history, etc.)
- **REQ_T31-T33_Interface_Communication_03**: A mechanism SHOULD allow the modelling process to query missing data needed for validation of the models

6.5.3 Deployment/reconfiguration-decision interface (T3.2/T3.3)

The Decision module sends an order to the *Deployment and Reconfiguration* module to select the reconfiguration actions[4].

Requirement identification: **T32-T33_Interface_Orders**

The following points will specify the components of the orders from the decision module to the Deployment and Reconfiguration module.

- **REQ_T32-T33_Interface_Orders_01**: The format of the orders SHALL be normalized.
- **REQ_T32-T33_Interface_Orders_02**: The Deployment and Reconfiguration module SHALL receive identified configuration plans from the Decision module.
- **REQ_T32-T33_Interface_Orders_03**: The order SHOULD include configuration management information.

Note: Configuration management should include:

- ✓ Versioning of the configurations
- ✓ Rollbacks to a prior configuration
- ✓ Validity period
- ✓ Priority
- ✓ Criticality
- ✓ Risk

Requirement identification: **T32-T33_Interface_Communication**

The communication between both modules would be synchronous and asynchronous.

- **REQ_T32-T33_Interface_Communication_01**: An exchange protocol SHALL exist to define the communications between the Decision Module and the Deployment and Reconfiguration Module.
- **REQ_T32-T33_Interface_Communication_02**: Interface communications SHOULD be reliable.

Requirement identification: **T32-T33_Interface_Security**

The exchanged information is composed of 4 points to be protected:

1) Primitives belonging to the exchange protocols listed above.

2) The index of configuration to ensure the reconfigurability process can retrieve a suitable configuration

3) Plan and sequence

4) State of concerned components

- **REQ_T32-T33_Interface_Security_01**: The information received by the Deployment and Reconfiguration module SHOULD be ciphered.
- **REQ_T32-T33_Interface_Security_02**: A mechanism that ensures the integrity of the information exchanged SHALL exist.

[4] The workflow of the system will be described extensively in the future architecture documents.

Requirement identification: **T32-T33_Interface_Feedbacks**

The feedback process (a functional concept) represents the control of every action done by the Deployment and Reconfiguration module. It is necessary to ensure the dependability of the reconfigurability process.

As the Decision module pushes the orders to the Deployment and Reconfiguration Module, this latter shall inform the Decision module of the operations' outcomes (using the feedback process).

- **REQ_T32-T33_Interface_Feedbacks_01**: There SHALL be positive (success) or negative (failure) feedbacks for each reconfiguration action.
- **REQ_T32-T33_Interface_Feedbacks_02**: There SHOULD be a format to define the semantics of feedbacks information.
- **REQ_T32-T33_Interface_Feedbacks_03**: Feedbacks SHOULD be uniquely identified.
- **REQ_T32-T33_Interface_Feedbacks_04**[5]: There SHALL be a classification of outcomes of reconfiguration operations (what happened).
- **REQ_T32-T33_Interface_Feedbacks_05**: The feedback SHOULD mention the version of the installed configuration.
- **REQ_T32-T33_Interface_Feedbacks_06**: The feedback SHOULD mention the date of the installed configuration.

6.6 Interfaces with planning modules

6.6.1 Planning-Translation interface

This section describes the main requirements that should be satisfied by the interface between the *Planning* and *Translation*, *Deployment,* and *Decision* modules. This interface deals with how the data stored in *Planning* can be supplied to whichever other modules in the architecture which may require them.

Requirement identification: **WP2-T31_Communication**

- **REQ_WP2-T31_Communication_01**: The translator SHALL retrieve the target system's architecture and the formulated policies that are to be enforced.
- **REQ_WP2-T31_Communication_02:** The translator SHALL be able to process incremental updates of an operational plan that may result from remodelling of the dependability and security properties of the target system.
- **REQ_WP2-T31_Communication_03**: The translator SHALL be able to be configured for obtaining its expected input.

Requirement identification: **WP2-T31_Interface_Security**

- **REQ_WP2-T31_Interface_Security_01:** A mechanism SHOULD guarantee the identity the components involved in this interface.
- **REQ_WP2-T31_Interface_Security_02**: A mechanism to ensure the security of the information exchanged MAY exist between the Planning and Validation module and the Translation module.

[5] The feedback shall indicate either the success of the operations or the type of error (e.g.: bad version installed on the component, link to the device lost, unknown requested configuration, etc.)

6.6.2 Deployment/reconfiguration- Operational planning interface

There should be no direct interface between those modules since *Operational Planning* information should first be translated before being used by Deployment modules.

6.6.3 Decision-Operational planning interface

There is no direct interface from *Decision* module to *Planning*.

6.7 Interfaces with Self-healing modules

6.7.1 Translation module - Self-Healing interface

There should be no direct interface between those modules according to the Description of Work. However, for implementation reasons, we might consider in the future that event monitoring and serious incident detection could be translated and sent to the decision module.

6.7.2 Deployment/reconfiguration - Self-Healing interface

Interfaces with Self-healing have been defined in D4.1. We remind them hereafter:

- **REQ-T43-T32-Interface-Ask:** The Hot Reconfiguration module MAY ask the Fast Reaction module to enforce a fast reaction
- **REQ-T43-T32-Interface-Result**: The Fast reaction module SHALL send information[6] describing the result of the fast reaction to the Hot Reconfiguration module, in case it was triggered by it.
- **REQ-T43-T32-Interface-Priority**: There SHOULD be a mechanism between the hot reconfiguration and fast reaction (WP4) modules to handle possible conflicts between local fast reaction and reconfiguration orders managed by the hot reconfiguration module.

6.7.3 Decision module - Detection interface

This section details the requirements for Events and Alarms[7] (Serious Incidents) formats, protocols and non-functional requirements of this interface.

Capability: **T33-WP4-Interface**

Requirement identification: **T33-WP4-Interface_Events**

- **REQ_T33-WP4-Interface_Events_01**: The Decision module SHALL receive normalized events from the serious incident Detection module.
- **REQ_T33-WP4-Interface_Events_02**: The events received SHALL contain information about the type of the event source.
- **REQ_T33-WP4-Interface_Events_03**: The events received SHALL contain information about the event data.

[6] These data may include the reaction applied, the target system, the current configuration, the success, or cancel status

[7] Alarms and Events are defined in the DESEREC Glossary document.

- **REQ_T33-WP4-Interface_Events_04**: The Detection module MAY receive information about the sensor which performs the event detection.

Requirement identification: **WP4_Interface_Alarms**

- **REQ_T33-WP4-Interface_Alarms_01**: The Decision module SHALL receive normalized alarms from the serious incident detection module.
- **REQ_T33-WP4-Interface_Alarms_02**: The alarms received SHALL contain information about the impact on the system.
- **REQ_T33-WP4-Interface_Alarms_03**: The events received SHALL contain information about the alarm data.

Requirement identification: **WP4_Interface_Communication**

- **REQ_T33-WP4-Interface_Communication_01**: A communication protocol SHALL exist between Event and Serious incident detection modules and the Detection module.

Requirement identification: **WP4_Interface_Security**

- **REQ_T33-WP4-Interface_Security_01**: The information received by the Detection module SHOULD be encrypted.
- **REQ_T33-WP4-Interface_Security_02:** Specific mechanisms SHALL ensure the integrity of the information exchanged

7 Classification of the requirements

Based on the methodology proposed in 2.2, we propose the following classification of the requirements identified in the previous section:

Identification	Priority	Test method
REQ_Translation_Component_Groups_01	M	Analysis **(A)**
REQ_Translation_Component_Groups_02	M	A/ Demonstration **(D)**
REQ_Translation_Configurations_01	M	A/D
REQ_Translation_Configurations_02	M	A/D
REQ_Translation_Configurations_03	M	A/D
REQ_Translation_Constraints_01	M	Inspection **(I)**
REQ_Translation_Output_01	M	D
REQ_Deployment_Configuration_01	M	A/Test Points **(TP)**
REQ_Deployment_Configuration_02	R	D
REQ_Deployment_Configuration_03	R	D
REQ_Deployment_Configuration_04	R	D
REQ_Deployment_Performance_01	M	A
REQ_Deployment_Performance_02	O	D
REQ_Deployment_Monitoring_01	M	D
REQ_Deployment_Monitoring_02	R	D
REQ_Deployment_Security_01	M	I/D
REQ_Deployment_Security_02	M	I/D
REQ_Deployment_Audit_01	M	D
REQ_Reconfiguration_ServiceContinuity_01	M	D
REQ_Reconfiguration_Storage_01	M	D
REQ_Reconfiguration_Storage_02	M	D
REQ_Reconfiguration_Storage_03	M	D
REQ_Reconfiguration_Storage_04	R	D
REQ_Reconfiguration_Supervision_01	M	D
REQ_Reconfiguration_Communication_01	M	A
REQ_Reconfiguration_Communication_02	R	D
REQ_Reconfiguration_Handle_01	M	D
REQ_Reconfiguration_Handle_02	M	D
REQ_Reconfiguration_Handle_03	M	D
REQ_Reconfiguration_Handle_04	R	D

REQ_Reconfiguration_Handle_05	R	D
REQ_Reconfiguration_Handle_06	M	D
REQ_Reconfiguration_Handle_07	R	D
REQ_Reconfiguration_Feedback_01	M	D
REQ_Reconfiguration_Feedback_02	R	TP
REQ_Reconfiguration_Feedback_03	M	D
REQ_Reconfiguration_Feedback_04	M	D
REQ_Reconfiguration_Performance_01	R	TP
REQ_Reconfiguration_Audit_01	M	D
REQ_DecisionModule_Storage_01	M	D
REQ_DecisionModule_Storage_02	M	D
REQ_DecisionModule_Storage_03	R	D
REQ_DetectionModule_SystemView_01	R	D
REQ_DetectionModule_SystemView_02	R	D
REQ_DetectionModule_SystemView_03	R	D
REQ_DetectionModule_SystemView_04	R	D
REQ_DetectionProc_EventManagement_01	M	D
REQ_DetectionProc_EventManagement_02	R	D
REQ_DetectionProc_Pattern_01	M	A/D/TP
REQ_DetectionProc_Investigate_01	O	A/D
REQ_DetectionProc_Knowledge_01	R	D
REQ_DetectionProc_Knowledge_02	R	TP
REQ_DetectionProc_Performance_01	M	A/D
REQ_DetectionProc_Performance_02	R	I/D
REQ_DetectionProc_UnknownSequence_01	R	I
REQ_DetectionProc_Security_01	M	I
REQ_DetectionProc_Handle_01	O	A
REQ_DetectionProc_SystemView_01	R	D
REQ_ReactionSel_Mapping_01	M	A
REQ_ReactionSel_Mapping_02	O	A
REQ_ReactionSel_Mapping_03	M	I
REQ_ReactionSel_Mapping_04	O	D
REQ_ReactionSel_Mapping_05	R	D
REQ_ReactionSel_Monitoring_01	R	A
REQ_ReactionSel_Monitoring_02	M	D
REQ_ReactionSel_Audit_01	M	D
REQ_ReactionSel_Audit_02	M	I
REQ_UserInterface_Administration_01	R	D

REQ_UserInterface_Reaction_01	M	A/D
REQ_UserInterface_Reaction_02	M	D
REQ_UserInterface_Reaction_03	M	A/D
REQ_UserInterface_Reaction_04	R	A/D
REQ_UserInterface_Security_01	R	D
REQ_UserInterface_Security_02	R	I
REQ_UserInterface_Monitoring_01	R	A/D
REQ_UserInterface_Monitoring_02	R	A/D
REQ_UserInterface_Monitoring_03	R	D
REQ_UserInterface_Monitoring_04	M	D
REQ_UserInterface_Monitoring_05	M	A
REQ_UserInterface_Monitoring_06	R	D
REQ_UserInterface_Monitoring_07	M	D
REQ_UserInterface_Monitoring_08	M	D
REQ_UserInterface_DESERECInfrastructure_01	R	D
REQ_T31-T32_Storage_01	M	D
REQ_T31-T32_ConfRules_01	M	D
REQ_T31-T32_ConfRules_02	O	D
REQ_T31-T32_Management_01	M	D
REQ_T31-T32_Communication_01	M	D
REQ_T31-T32_Communication_02	M	D
REQ_T31-T32Translation_Security_01	R	I/D
REQ_T31-T33-Interface_Security_01	R	D
REQ_T31-T33-Interface_Security_02	O	D
REQ_T31-T33-Interface_Detection_01	M	A
REQ_T31-T33-Interface_Detection_02	M	A
REQ_T31-T33-Interface_Detection_03	M	D
REQ_T31-T33-Interface_Reaction_01	M	A
REQ_T31-T33-Interface_Reaction_02	M	D
REQ_T31-T33-Interface_Communication_01	M	A
REQ_T31-T33-Interface_Communication_02	M	D
REQ_T31-T33-Interface_Communication_03	R	D
REQ_T32-T33_Interface_Orders_01	M	I
REQ_T32-T33_Interface_Orders_02	M	TP
REQ_T32-T33_Interface_Orders_03	R	I
REQ_T32-T33_Interface_Communication_01	M	I
REQ_T32-T33_Interface_Communication_02	R	TP
REQ_T32-T33_Interface_Security_01	R	I

REQ_T32-T33_Interface_Security_02	M	A
REQ_T32-T33_Interface_Feedbacks_01	M	D
REQ_T32-T33_Interface_Feedbacks_02	R	I
REQ_T32-T33_Interface_Feedbacks_03	R	I
REQ_T32-T33_Interface_Feedbacks_04	M	A/D
REQ_T32-T33_Interface_Feedbacks_05	R	D
REQ_T32-T33_Interface_Feedbacks_06	R	D
REQ_WP2-T31_Communication_01	M	D
REQ_WP2-T31_Communication_02	M	D
REQ_WP2-T31_Communication_03	M	D
REQ_WP2-T31_Interface_Security_01	R	A
REQ_WP2-T31_Interface_Security_02	O	I
REQ_T43-T32-Interface_Ask	O	I
REQ_T43-T32-Interface_Result	M	I
REQ_T43-T32-Interface_Priority	R	I
REQ_T33-WP4-Interface_Events_01	M	I
REQ_T33-WP4-Interface_Events_02	M	I
REQ_T33-WP4-Interface_Events_03	M	I
REQ_T33-WP4-Interface_Events_04	O	I
REQ_T33-WP4-Interface_Alarms_01	M	I
REQ_T33-WP4-Interface_Alarms_02	M	I/D
REQ_T33-WP4-Interface_Alarms_03	M	I/D
REQ_T33-WP4-Interface_Communication_01	M	A/TP
REQ_T33-WP4-Interface_Security_01	R	I/D
REQ_T33-WP4-Interface_Security_02	M	I/D

8 Conclusion

This document presented the requirements for Decision and Reconfiguration modules (translation, deployment and hot reconfiguration, and decision). The requirements expressed in this deliverable have been derived from the project's objectives and from general constraints that were briefly reminded respectively in sections 2 and 5. Then, we have classified the requirements based on their priority, timeframe, test method, and link to scenarios.

We have expressed 131 requirements including 72 internal and 59 external requirements (with other Decision and Reconfiguration modules or with modules developed by other external modules such as Fast Self-Healing ones). Moreover, 34 requirements were defined as mandatory and short term among a complete list of 75 mandatory requirements.

These requirements will serve as a basis for the next steps which will consist in defining the products architecture and specifications for all the modules. This task will be achieved in D3.2.