Interoperable Trust Assurance Infrastructure

Specification and design of the secure interoperability framework and tools (First version)

Type (distribution level)	Public *(according to DoW)*
Contractual date of Delivery	30-04-2013
Actual date of delivery	30-04-2013
Deliverable number	**D4.2.1**
Deliverable name	**Specification and design of the secure interoperability framework and tools - first version.**
Version	V1.0
Number of pages	66
WP/Task related to the deliverable	WP4/T4.2
WP/Task responsible	*MI/MI*
Author(s)	Wissam Mallouli, Edgardo Montes de Oca, Bachar Wehbi (MI), Lidia Fuentes, Monica Pinto, José Miguel Horcas (UMA), Jorge Bernard Benabé, Juan Manuel Marín Pérez (UMU), Samiha Ayed, Nora Cuppens, Frederic Cuppens, Khalifa Toumi, Ana Cavalli (IMT), Eszter Kerezsi (SL)
Partner(s) Contributing	*ALL*
Document ID	*INTER-TRUST-T4.2-MI-DELV-D4.2.1-SpecDesSecInterFram&Tools-*

SEVENTH FRAMEWORK PROGRAMME

🖫	First-V1.00
Abstract	*This deliverable presents a first version of the specification and design of the secure interoperability framework and tools. It provides the specification of the framework to deploy the security requirements, how the requirements are interpreted, how they are integrated into the applications, the specification of the secure interoperability tools, and the monitoring and testing tools. The chosen dynamic AOP framework, as well as the way it will be used to weave the security requirements interpreter, are also specified in this document.*

Executive summary

This deliverable is the main output of Task 4.2 (Specification/implementation of modules, aspects and tools in a security requirements interpreter framework) in WP4 (Techniques & tools for secure interoperation enforcement and supervision). As part of task 4.2, we have identified the main building blocks of the INTER-TRUST framework based on the requirements defined in WP2 (D2.1.1); the preliminary results of WP2 task 2.3 that focuses on main functionalities in the INTER-TRUST framework architecture; and, the result of document D4.1 that describes the Aspect Oriented Programming (AOP) framework selected in the context of INTER-TRUST case studies. This deliverable provides a first detailed specification of the different architecture components including modules, aspects and tools and their potential interactions. It also presents partners' existing tools and their adaptation to fit the INTER-TRUST requirements and constraints. The results of the architecture components specification is reported in the present deliverable and will constitute a first input for document D4.3.1 that will provide the implementation of the INTER-TRUST framework, as well as for WP5 that will evaluate it in the context of the three INTER-TRUST case studies (V2V, V2I communications and E-voting system).

This document is organized in 10 sections (excluding introduction and conclusion sections). The second section provides a **high level description of the INTER-TRUST global framework** which constitutes the main thread linking the nine other sections (from 3 to 11). The main objective of this framework is to support trustworthy services and applications in heterogeneous networks and devices based on the enforcement of interoperable and changing security policies. It addresses the needs of developers, integrators and operators to develop and operate systems in a secure trusted manner dictated by negotiated security policies through dynamic SLAs. Each main component (i.e. module, aspect or tool) of the INTER-TRUST framework is presented in a dedicated section where we describe its functionalities, external and internal data structures, public interfaces and interaction with other components. This includes the **security editor tools** to specify dynamic and contextual security policies and **the trust negotiation module** that allow different interacting parties to define a common security policy through the use of predefined negotiation models - this will allow the different parties to collaborate in a trusted manner; **the security policy interpreter** that will be woven into the applications to interpret the negotiated policy; **the aspects Generation module** to dynamically generate aspects to be woven or unwoven by **the aspect Weaver module**; **the monitoring and testing modules** that serve to stimulate the system by injecting code for active and fuzz testing (for the development/test phases) and capture application events to analyse them for security checking; **the reaction module** that is in charge of performing the necessary protection and mitigation strategies; and finally three **stand-alone monitoring and testing tools** will be used to supervise and verify that the system works as expected: MMT monitoring tool, TESTGEN-IF active testing tool and FLINDER fuzz testing tool.

The goal of this document is not to perform an exhaustive specification of the INTER-TRUST components, but only to define their main features, interfaces and interactions with other components. The first implementation of these components, the delivery of the consolidated INTER-TRUST framework architecture (outcome of task 2.3) and the first evaluation planned in WP5 will allow us to refine this specification and provide a second, detailed version of the document (D4.2.2) incorporating the first feedbacks of the integration in the INTER-TRUST case studies.

Table of contents

1 Introduction

1.1 Scope of the document

The main objective of the INTER-TRUST project is to develop a dynamic and scalable framework[1] to support security mechanisms that allow a secure interoperation between different parties or systems, and to be able to dynamically adapt the security rules that drive this interaction according to changes in the running environment. This framework aims to provide formal models, methods, techniques and tools for developers, testers and integrators to produce more secure applications that are able to adapt their behaviour, using AOP mechanisms, both to the security policies that are provided to them and to the "context" in which these applications operate. It also aims to provide run-time components that monitor the application during operation and react, in the case of security policy violations, by injecting ad-hoc code, using AOP mechanisms to protect the inter-operating system against attacks. Active and fuzz testing will complete the approach by performing conformance and robustness testing. Active testing will be used to verify that the implementation of security policies respects the interoperability requirements. Fuzz testing will be used to check the robustness of the implementation against malicious input.

As this document is the first version of the specification and design of the INTER-TRUST secure interoperability framework, there is rather difference in specification details for some modules and tools. The second version of the document will recover this issue mainly after the delivery of the consolidated framework architecture due to month 11 (Task 2.3 in WP2).

1.2 Applicable and reference documents

This document refers to the following documents:

- D2.1.1 Requirements specification first version (delivered on month 4 in WP2). The INTER-TRUST framework requirements are taken into account in order to define to INTER-TRUST framework components that target the specified requirements.
- D4.1 Analysis and selection of the AOP framework (delivered on month 5 in WP4). Provides a detailed analysis of the existing AOP frameworks and selects the one that is best suited to answer the INTER-TRUST requirements. This AOP framework provides the basis of the whole INTER-TRUST solution.
- D2.3 INTER-TRUST approach and framework specification first version (to be delivered on month 11 in WP2). This document will specify of the framework architecture including scenarios and main use-cases.
- D4.3.1 Implementation of the secure interoperability framework and tools first version (to be delivered on month 11 in WP4) Implementation of the integrated framework to deploy the security requirements.
- D5.1.1 Integration report first version (to be delivered on month 10 in WP5). This report describes the integration of the framework in different INTER-TRUST case studies and gives the configuration and installation procedures for the case study evaluations.

[1] By framework we mean a set of methods, libraries, tools, rules and conventions that allow developing, deploying and operating applications with the capability of assuring secure interoperability and adaptability.

1.3 Revision History

Version	Date	Author	Description
V0.1	01-14-2013	Wissam Mallouli	Creation of the document and proposition of a table of content
V0.2	02-26-2013	Lidia Fuentes	First contribution of UMA
	02-26-2013	Jorge Bernard Benabé	First contribution of UMU
	03-18-2013	Wissam Mallouli	Revision of the table of content
	03-23-2013	Monica Pinto	Second contribution of UMA
	03-27-2013	Khalifa Toumi	First contribution of IT
	03-28-2013	Jorge Bernard Benabé	Second contribution of UMU
	03-30-2013	Samiha Ayed, Nora Cuppens, Frederic Cuppens	Second contribution of IT
	04-01-2013	Wissam Mallouli	First contribution of MI – MMT tool description and adaptation
V0.3	04-02-2013	Wissam Mallouli	First integrated version of the document
	04-04-2013	Eszter Kerezsi	First contribution of SL
	04-05-2013	Lidia Fuentes	Third contribution of UMA
	04-05-2013	Samiha Ayed, Khalifa Toumi	Third contribution of IT
	04-05-2013	Wissam Mallouli	Second contribution of IT – Monitoring aspect module, executive summary and introduction
V0.4	04-05-2013	Wissam Mallouli	Second integrated version of the document
	04-08-2013	Juan Manuel Marín Pérez	Revision of section 5 (UMU)
	04-12-2013	Eszter Kerezsi	Final contribution of SL

	04-12-2013	Lidia Fuentes	Revision of Sections 6 and 7 (UMA)
V0.5	04-15-2013	Wissam Mallouli	Consolidation of the document and conclusion
	04-26-2013	Daniel Thiemert	Document review (UoR)
	04-29-2013	Eszter Kerezsi	Document review (SL)
V1.0	04-30-2013	Wissam Mallouli	Final version of the document

1.4 Notations, abbreviations and acronyms

AJDT: AspectJ Development Tool

AOSD: Aspect-Oriented Software Development

AOP: Aspect-Oriented Programming

AJC: AspectJ Compiler

AS: Application Server

CBSE: Component-Based Software Engineering

CLI: Common Language Infrastructure

IDE: Integrated Development Environment

IoC: Inversion Of Control

JAAS: Java Authentication and Authorization Service

JVM: Java Virtual Machine

LTW: Load-Time Weaving

OOP: Object-Oriented Programming

RE: Regular Expression

UML: Unified Modelling Language

2 INTER-TRUST framework overview

The deliverable D2.1.1 provided an initial set of functional and non-functional requirements for the INTER-TRUST framework. These requirements have been elicited following the UI-REF methodology [15] taking into account target domains (Vehicle to Infrastructure and Vehicle to Vehicle communications and electronic voting systems) as well as standards, gap analysis and market watch. The study of these requirements allowed us to define the INTER-TRUST framework functionalities that target the following main requirements:

1. The autonomous **definition of the security policies**, with the use of languages to model security rules, negotiation mechanisms and threats.

2. The separation of security concerns from applications concerns, with the use of Aspect Oriented Programming techniques to **deploy the security requirements** at run-time.

3. The **dynamic adaptation to changes** in the environment context, with the use of Aspect Oriented Programming techniques to react by activating aspects to implement protection and mitigation strategies.

4. The verification of the system's **conformance and robustness** with the use of active and **Fuzz Testing**.

5. The **detection of non-compliance** with the security requirements, threats and other context information, with the use of supervision techniques based on monitoring and the reaction according to the mitigation strategy.

These concepts are the basis for the first design of the INTER-TRUST framework presented in Figure 1, integrating the different framework functionalities and modules. In the following, we present the role of each module of the INTER-TRUST framework:

Security editor tools: These tools allow to model security policies, threats, interoperability constraints, negotiations models and contracts. Existing languages and editing tools will be adapted and used according to the results of WP3 task 3.1 "Description of models for the specification of secure interoperability policies". This security policy supporting interoperability may generally include: access control requirements, permissions and prohibitions; usage control requirements, obligations to respect; it may correspond to complex requirements and may comprise temporal deadline conditions that specify what happens in the case of violation of any of the contracts, e.g., sanctions that are triggered when a violation is detected.

Negotiation module: Two types of negotiations are considered in this module. The first one aims to build security policies supporting interoperability. It can be viewed as a set of contracts, negotiated between two (or more) different entities that are applied to control and regulate their interoperation. Challenges related to this module that will be addressed in the INTER-TRUST project are: (1) defining the modelling formalism to express a certain number of distinct security policies, security Service Level Agreements (SLAs) and relationships beyond those expressed by existing formalisms; and (2), the dynamic introduction of negotiated and deployed security SLAs. The second involves trust negotiation. It is an approach for gradually establishing trust between interacting parties, strangers to each other. It is based on a bilateral disclosure of information by the negotiating parties, called digital credentials, used to prove their trustworthiness. The essential elements of a trust negotiation process are digital credentials, disclosure policies and negotiation strategies. Only the trust negotiation module will be specified in this first version of the document.

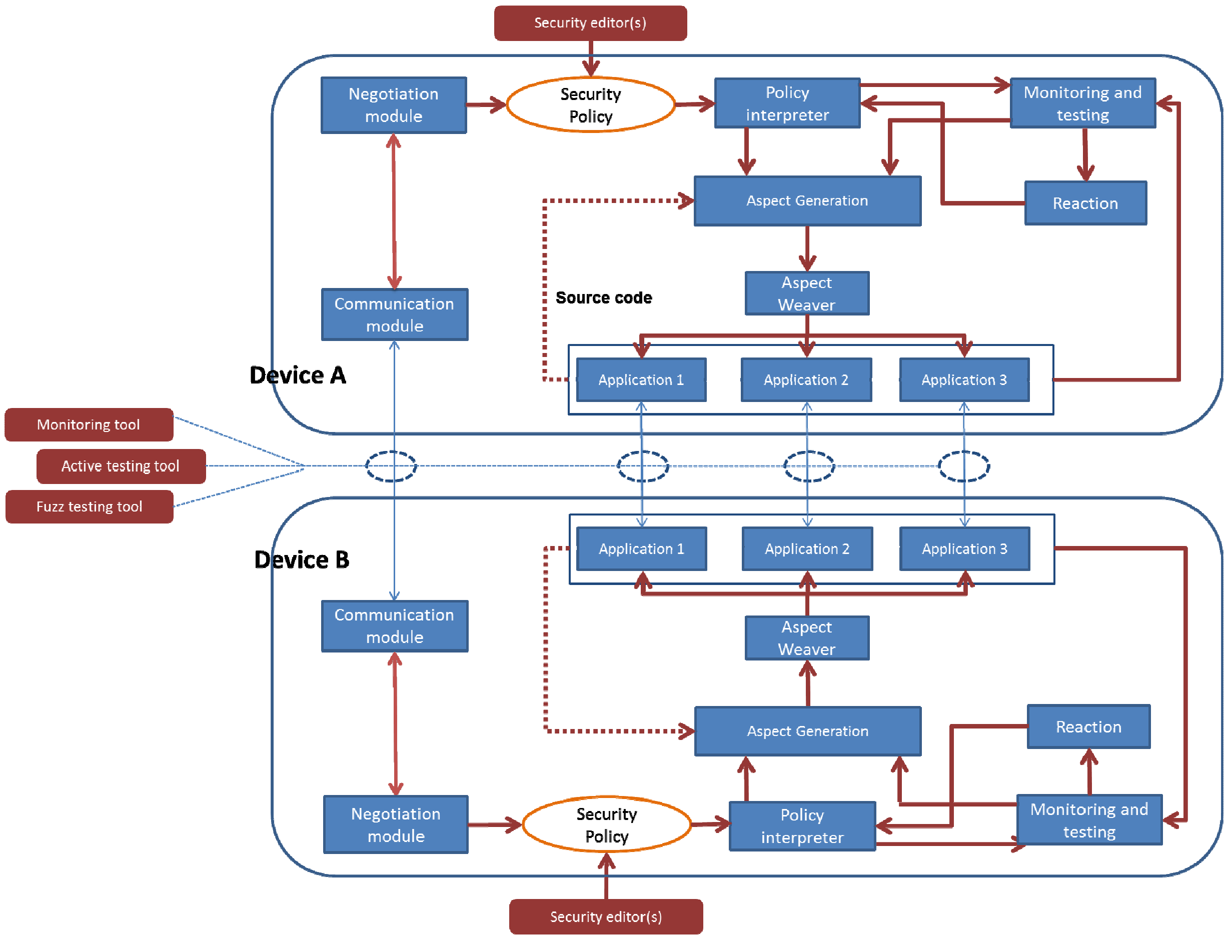


Figure 1. The INTER-TRUST framework global architecture

Policy interpreter: This module will interpret the negotiated policies. It is the module where the decision on the already negotiated security policy will be taken. It plays the role of Policy Decision Point (PDP). A PDP examines the request, retrieves policies that are applicable to this request, and determines whether access should be granted according to the rules for evaluating policies.

Aspects Generation module: This module will allow to dynamically generate aspects to be woven, based on the negotiated policy. This includes code to obtain context awareness. It is divided into two parts: aspect pattern generation and aspect concrete generation.

Aspect Weaver module: This dynamic module will allow weaving and un-weaving aspects. The policies are contextualised depending on the situation where they are to be used (e.g. the specific parties that need to interact, time limits when they are valid, etc.). We show in the deliverable D4.1.1 how AOP can be used to dynamically weave and un-weave security requirements "on the fly" into applications and protocols used by parties that need to interoperate. To illustrate this, a simple example is called action modification, where a function call is replaced by another function call (e.g. secureSend (x, policy_A) by secureSend (x, policy_B)). This type of dynamic weaving will be done with resource efficiency in mind.

Monitoring and testing aspect modules: These modules will serve to: (1) stimulate the system by injecting code or input messages for active and fuzz testing (Inject aspect, for the development/test and operation phases); (2) capture application events to generate traces that can be used by the Monitoring tool (for the development/test and operation phases); (3) detect the non-compliance of security requirements (during the operation phase); and, (4) generate warnings or alarms that will provoke the reaction module and enforce secure interoperability (during the operation phase).

Reaction module: This module will be in charge of performing the necessary protection and mitigation strategies. The security requirements will define protection and reaction strategies. In this way it will be possible to increase the reliability and trust of the proposed security rules; and, the adaptability of the system to new sets of malicious behavior and threats.

Stand-alone monitoring and testing tools are used to complete the framework and verify that the system works as expected. An existing monitoring tool (provided by MI) will be adapted to detect any vulnerabilities and abnormal behavior. During the testing and operation phases it will serve to observe the behavior of the system; detect security vulnerabilities; verify that the negotiated contracts are respected; verify that the sharing of information and the delegation of processing is carried out securely; and, trigger reaction strategies. Other active and fuzz testing tools (provided by IT and SL) will be adapted to verify that security policies are respected in dynamic systems during development/testing phases. Combined with the monitoring tool, it will be possible to stimulate and attack the system under test and detect vulnerabilities.

The INTER-TRUST framework, with its different modules, addresses all the steps of modelling and deployment of security requirements, namely: (1) security policy specification; (2) security policy deployment; (3) security component configuration; and, (4) security policy redeployment in case of changes in the environment, in particular when an intrusion is detected.

Aspect Oriented Programming (AOP) will be used by the INTER-TRUST framework to add/implement security sub-concerns (i.e. availability, authentication, access control, integrity, encryption, enrolment, etc.) to application components. Thus, it can be used to weave security-related concerns and properties “on the fly” so that each party can dynamically adapt its behaviour to the negotiated security contracts. To complement the approach, formal-based monitoring techniques will be used as a supervision technique to detect changes in the environment and to check whether the involved parties actually respect the negotiated contracts. This information will be fed back to the framework, and the involved parties, so that they can use it to adapt themselves to new threats and constraints that may arise. In this way, applications will be able to have a global understanding of risks and automatically react when needed. This approach represents a generic solution that can be applied to many types of pervasive applications.

3 Security editor tools

In this section, four security policy editor tools are presented: MotOrBAC editor, UMU XACML editor, XACML Web editor and UMU policy console. These tools will be adapted in the context of INTER-TRUST project to fit the models constraints defined in WP3.

3.1 MotOrBAC editor

3.1.1 Overview

OrBAC [16] stands for Organization Based Access Control language. It is an access and usage control model based on first order logic that allows an organization to express its security policy including contextual rules. For this purpose, OrBAC defines two abstraction layers. The first one is called abstract layer and describes a rule as a *role* having the permission, prohibition or obligation to perform an *activity* on a *view* in a given *context*. A *view* is a set of objects to which the same security rules apply. A *role* is set of users with similar privileges and an *activity* considers a set of actions with similar properties. The second layer is the concrete one. It is derived from the abstract level and grants permission, prohibition or obligation to a *user* to perform an *action* on an *object*.

The *OrBAC* model has an associated tool called MotOrBAC that was developed by Institut Télécom (IT) to help to design and implement security policies using the Or-BAC model. The current versions of this tool can design, upload and store security policies and simulate them. The policy simulation can be used to verify the consistency of a security policy. The tool can also detect potential conflicts and help the designer eliminate them. MotOrBAC exists in two versions. The first version is completely free and is distributed under GPL license. The second version, newer and more functional and actively developed, is partially open source and is distributed under Mozilla license. Unlike the first version, MotOrBAC V2 is implemented entirely in Java. It uses an API that has been specially developed to incorporate features of the implementation of the OrBAC model in existing or under development applications. This API, the OrBAC API, is not open source but can be requested on the official Web site of OrBAC[2].

3.1.2 MotOrBAC architecture

The MotOrBAC architecture is defined in Figure 2.

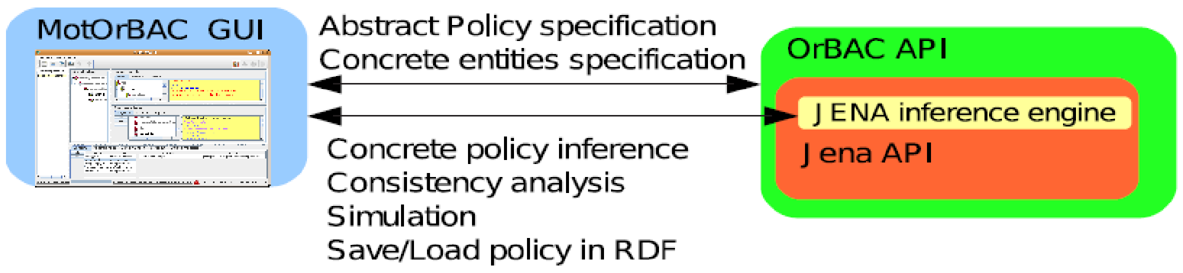


Figure 2. MotOrBAC tool architecture

[2] http://www.orbac.org/.

MotOrBAC uses the *OrBAC* application programming interface (API) to manage the policies displayed in the graphical user interface (GUI). The *OrBAC* API can be used to programmatically create *OrBAC* policies. The concrete security policy inferred by MotOrBAC can be translated to configure a security component such as a firewall for example. Another solution to enforce an *OrBAC* security policy is to use the *OrBAC* Java API, on top of which MotOrBAC is built. This API uses the *Jena* java library[3] to represent an *OrBAC* policy as a RDF graph. It can be used to load MotOrBAC RDF policies and interpret them, i.e. access control requests can be made on a loaded policy. *Jena* features an inference engine which is used by the *OrBAC* API to infer the concrete policies and the conflicts. When an *OrBAC* RDF policy is loaded by the API, the concrete policy can be inferred and stored in memory. An instance of the *OrbacPolicy* java class which encapsulates an *OrBAC* policy uses a cache of concrete security rules to enhance the performances when the policy is queried. Contexts are evaluated upon a query; this feature is actually used in the MotOrBAC simulation tool to show the contexts state. The contexts implementation can be easily extended in order to interface the API with other applications and add new types of contexts.

Integrating the OrBAC Java API into a Java application can be done without modifying the application source code. Aspect Oriented Programming (AOP) can be used to separate security concerns from other concerns relative to the application.

3.1.3 Functionalities

MotOrBAC can be used to perform several tasks on *OrBAC* security policies:

- Edit policies: the administrator can create the abstract entities he/she needs (organizations, sub-organizations, roles, activities, views, contexts) and the abstract security policies. Hierarchies defined for these concepts are also defined. These different concepts and policies can be expressed and defined through a graphical interface. Figure 3 shows the definition of hierarchies as a graphical tree. Many types of security rules can be specified using MotOrBAC. We can define permissions, prohibitions and obligations. Figure 3 shows these different types. Many types of contexts are available to express conditions of rules activation. Each type of context define a specification language like Beanshell language (set of Java language), Prova (set of Prolog).

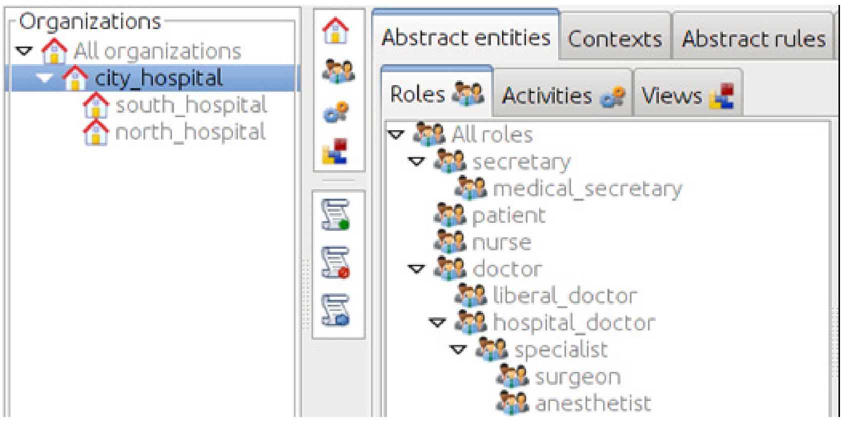
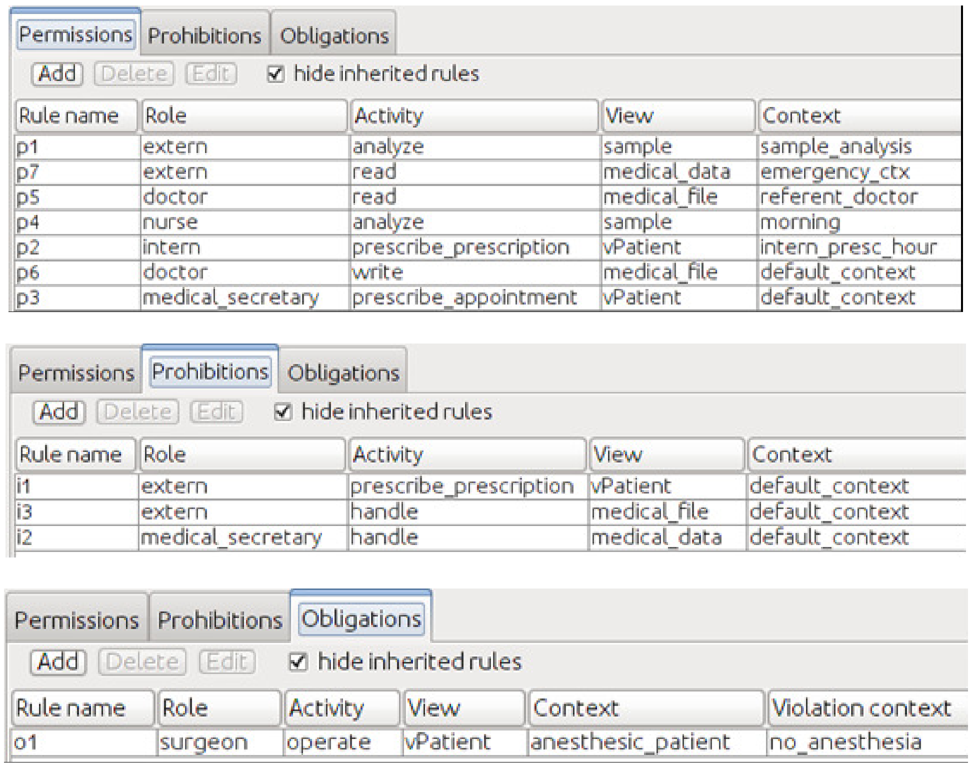


Figure 3. Graphical hierarchy representation in MotOrBAC

[3] http://jena.apache.org/

- Policy simulation: after having specified concrete entities (subjects, actions and objects), the concrete policy can be inferred. Subjects, actions and objects can have attributes. The simulation interface is presented in Figure 5.



☑ hide inherited rules

Rule name	Role	Activity	View	Context
p1	extern	analyze	sample	sample_analysis
p7	extern	read	medical_data	emergency_ctx
p5	doctor	read	medical_file	referent_doctor
p4	nurse	analyze	sample	morning
p2	intern	prescribe_prescription	vPatient	intern_presc_hour
p6	doctor	write	medical_file	default_context
p3	medical_secretary	prescribe_appointment	vPatient	default_context

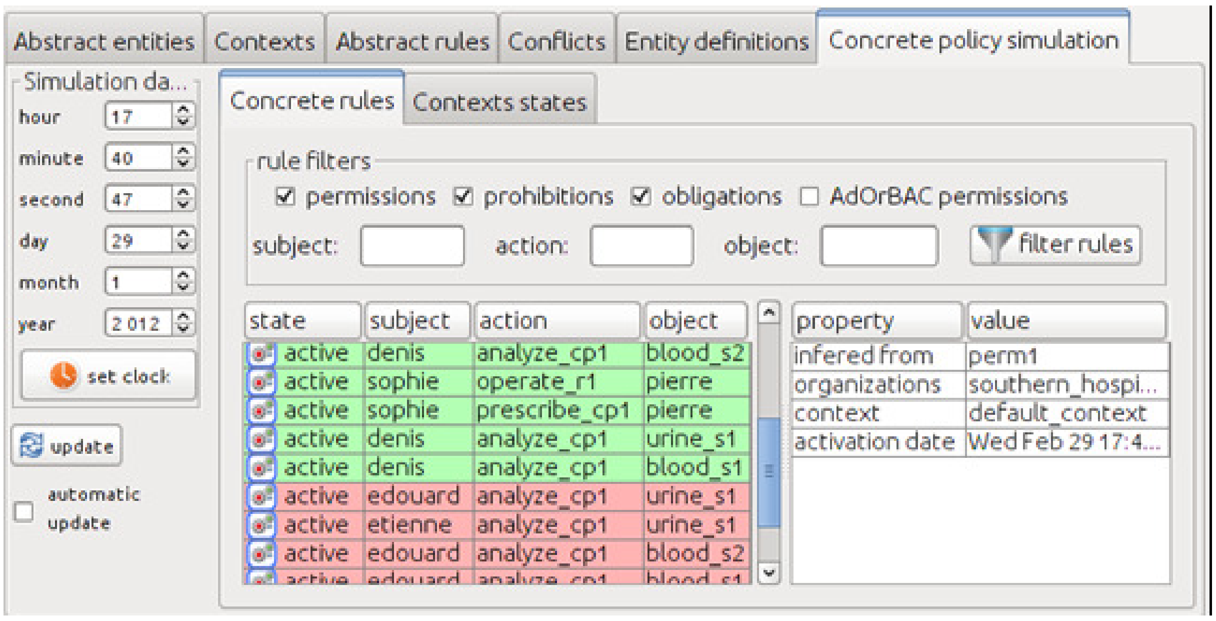
☑ hide inherited rules

Rule name	Role	Activity	View	Context
i1	extern	prescribe_prescription	vPatient	default_context
i3	extern	handle	medical_file	default_context
i2	medical_secretary	handle	medical_data	default_context

☑ hide inherited rules

Rule name	Role	Activity	View	Context	Violation context
o1	surgeon	operate	vPatient	anesthesic_patient	no_anesthesia

Figure 4. Example of abstract rules



hour	17
minute	40
second	47
day	29
month	1
year	2 012

☐ automatic update

☑ permissions ☑ prohibitions ☑ obligations ☐ AdOrBAC permissions

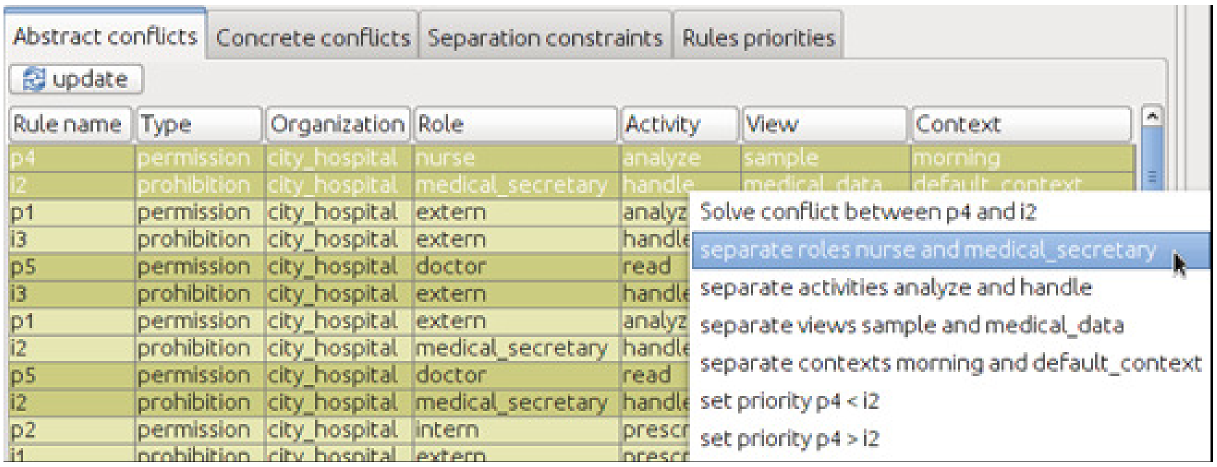
subject: action: object:

state	subject	action	object
active	denis	analyze_cp1	blood_s2
active	sophie	operate_r1	pierre
active	sophie	prescribe_cp1	pierre
active	denis	analyze_cp1	urine_s1
active	denis	analyze_cp1	blood_s1
active	edouard	analyze_cp1	urine_s1
active	etienne	analyze_cp1	urine_s1
active	edouard	analyze_cp1	blood_s2
active	edouard	analyze_cp1	blood_s1

property	value
infered from	perm1
organizations	southern_hospi...
context	default_context
activation date	Wed Feb 29 17:4...

Figure 5. Simulated of concrete policy

- Policy consistency verification: abstract conflicts between abstract rules can be detected. Once abstract conflicts have been detected, MotOrBAC is able to suggest the administrator some solutions to solve them. Figure 6 presents an interface of possible managements of these conflicts.

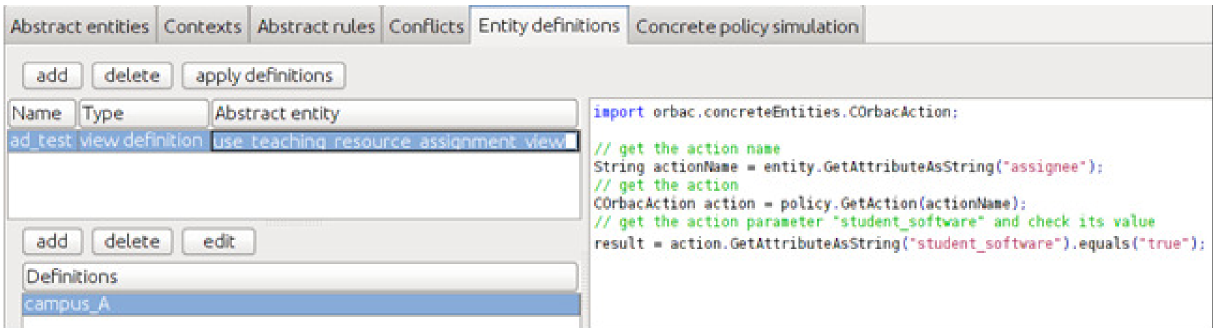


Rule name	Type	Organization	Role	Activity	View	Context
p4	permission	city_hospital	nurse	analyze	sample	morning
i2	prohibition	city_hospital	medical_secretary	handle	medical_data	default_context
p1	permission	city_hospital	extern	analyz		
i3	prohibition	city_hospital	extern	handle		
p5	permission	city_hospital	doctor	read		
i3	prohibition	city_hospital	extern	handle		
p1	permission	city_hospital	extern	analyz		
i2	prohibition	city_hospital	medical_secretary	handle		
p5	permission	city_hospital	doctor	read		
i2	prohibition	city_hospital	medical_secretary	handle		
p2	permission	city_hospital	intern	prescr		
i1	prohibition	city_hospital	extern	prescr		

Solve conflict between p4 and i2
separate roles nurse and medical_secretary
separate activities analyze and handle
separate views sample and medical_data
separate contexts morning and default_context
set priority p4 < i2
set priority p4 > i2

Figure 6. Conflict management interface

- Administrative rights management: the administrative rights of a subject or a role can be specified in order to decentralize the policy administration. MotOrBAC is able to express the administration policy using the same formalism. It can be used to specify the administration policy, each AdOrBAC policy associated to each policy. AdOrBAC model implemented in API OrBAC allows a decentralized administration of the policy. We can for example control creating abstract entities, creating rules, etc. Figure 7 represents the definition of a view by specifying the insertion of objects in the «use_teaching_resource_assignment_view» sub-view of the view «activity_assignment_view».



Name	Type	Abstract entity
ad_test	view definition	use_teaching_resource_assignment_view

Definitions
campus_A

```
import orbac.concreteEntities.COrbacAction;

// get the action name
String actionName = entity.GetAttributeAsString("assignee");
// get the action
COrbacAction action = policy.GetAction(actionName);
// get the action parameter "student_software" and check its value
result = action.GetAttributeAsString("student_software").equals("true");
```

Figure 7. View definition

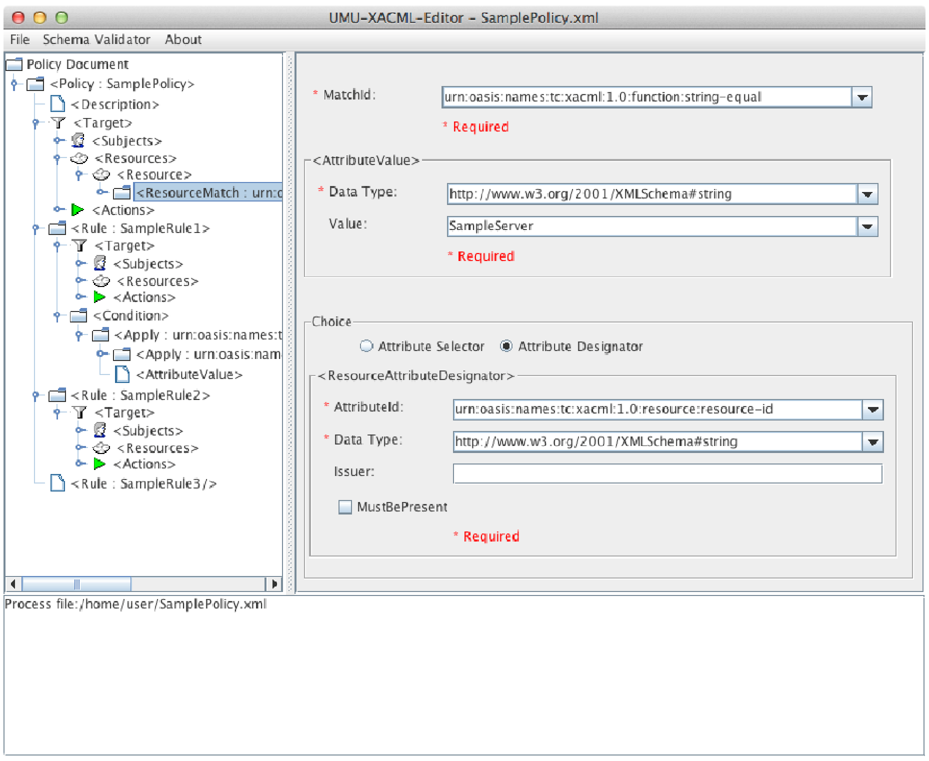
3.2 UMU XACML Editor

The UMU XACML Editor is a graphical console tool developed in Java that enables the definition of access control policies in XACML. This tool has been developed by the University of Murcia as part of a research effort on using different XML-related standards and technologies, such as XACML and SAML, to solve different scenarios on authentication and authorization of users and devices when accessing a network and the resources existing in that network.

The console is actually intended to cover XACML 2.0 standard policies as defined by the OASIS eXtensible Access Control Markup Language (XACML) TC [1]. It provides a partial coverage of the standard, allowing the administrator to define all aspects and parameters of the policy as specified by XACML. The console enables the creation, loading and saving of XACML policies in XML files. It also supports policy validation against the schema.

The tool is available in sourceforge and can be downloaded under the GNU LESSER GENERAL PUBLIC LICENSE Version 2.1.

Figure 8 shows the main view of the UMU-XACML Editor. The main frame shows three working areas: the top-left area contains a tree representation of the policy document, including different policies and policy sets that can be browsed; the main area (top-right) shows the details of the selected tree node and enables policy definition and edition; the bottom panel shows output messages about the editor operation.



* MatchId: urn:oasis:names:tc:xacml:1.0:function:string-equal
* Required

* Data Type: http://www.w3.org/2001/XMLSchema#string
Value: SampleServer
* Required

( ) Attribute Selector (●) Attribute Designator

* AttributeId: urn:oasis:names:tc:xacml:1.0:resource:resource-id
* Data Type: http://www.w3.org/2001/XMLSchema#string
Issuer:
☐ MustBePresent
* Required

```
Process file:/home/user/SamplePolicy.xml
```

Figure 8. UMU XACML Editor main view

This tool can be used by INTER-TRUST for access control policy definitions in XACML by administrators. It is initially designed to support XACML 2.0, but it could be adapted to support the latest XACML version 3.0 if needed. This editor tool provides a powerful interface that covers almost all aspects of XACML policy definitions. It can be combined with the XACML Web Editor tool, which provides a simplified and more user-friendly interface for basic XACML policy definitions.

3.3 XACML Web Editor

The XACML Web Editor is an access control editor tool that is provided via a Web interface. The tool has been developed as part of the SEMIRAMIS EU FP7 project [2] to provide a means for users to define policies that control the release of their personal information. The editor has been developed in Java and it is delivered as a Web application to be deployed in a Java Servlet container such as Apache Tomcat.

The editor tool generates access control policies in XACML 2.0. The interface is designed to be user friendly and not to expose every detail of XACML complexity. Thus, it does not provide the whole set of XACML features. It is able to manage XACML policies that may reside either in the server disk as

XML files or within an eXist XML database. It also allows importing and exporting XACML policies to the client device.

The console mainly consists of a main menu and three administration views: policy management, resource management and subject management. Subjects and resources are defined in their corresponding views to be used by policies. For each resource, its corresponding actions are also specified in the resource management view. Figure 9 shows the policy management view. The main frame of this view shows three different areas. The left area shows the list of policies, while on the right side the rules for the selected policy are edited. For each policy, different rules can be defined (bottom-right area) and the parameters of the selected rule (resources, subjects and actions) are also specified (up-right area).

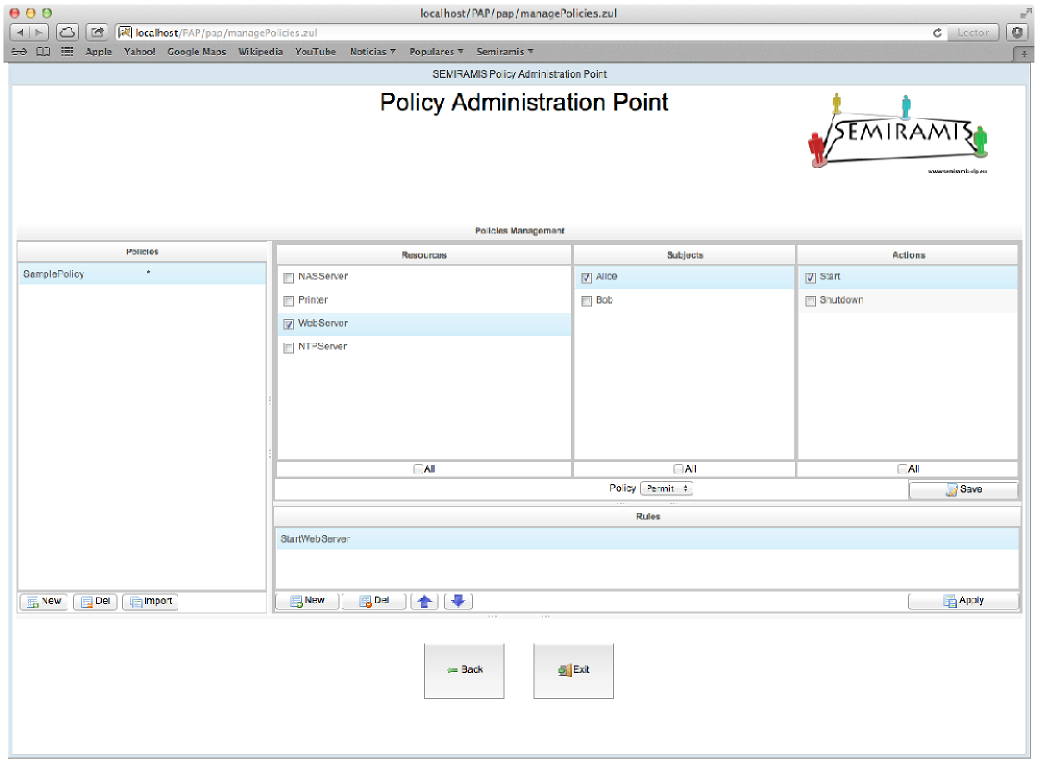


Figure 9. Policy Management view of the XACML Web Editor

The XACML Web Editor provides a user-friendly access control policy editor that can be accessed remotely from any web browser. This tool can be adopted by INTER-TRUST to enable easy and user-friendly definition of XACML policies for remote services through a web interface. Thus, access control policies could be defined from any device with web browsing capabilities. It can be complemented by the UMU XACML Editor tool in case specific aspects or parameters of the policy need to be tuned by the administrator.

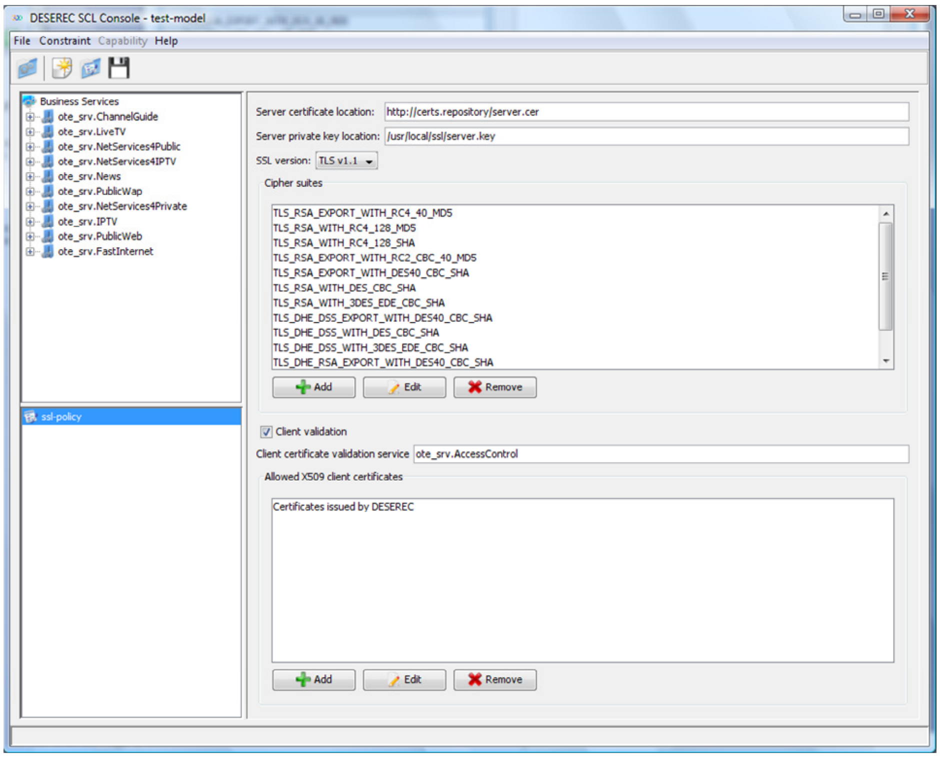
3.4 UMU Policy Console

The Policy Console is a graphical policy editor tool developed in Java under the scope of the DESEREC EU FP6 project [3]. It enables the administrator to create the security policies that will be used for generating the operational configurations of the system.

Policy definitions in the console are driven by the services provided by the system. Each service is considered to be comprised by a number of individual service components that, in turn, can be assigned with different kinds of policies. This provides an abstraction layer over the managed physical system that enables the definition of security policies. The console is a model-based tool. It uses an XML model representing the services and service components of the managed system.

The console is a tool for creating an XML-based security policy model based on an extension of the Security Policy Language (SPL) developed in the POSITIF FP6 EU project [4]. The model specifies which the security configuration policies for a given system are, and how the policies defined in it relate to the services and service components of such a system.

Figure 10 shows the main view of the console. There are three well-defined areas in the frame: one for browsing through the services and service components (top-left area), a bigger one for creating the policies (right area) and a last one for browsing through the created policies and assigning them to service components (bottom-left area).



Business Services
- ote_srv.ChannelGuide
- ote_srv.LiveTV
- ote_srv.NetServices4Public
- ote_srv.NetServices4IPTV
- ote_srv.News
- ote_srv.PublicWap
- ote_srv.NetServices4Private
- ote_srv.IPTV
- ote_srv.PublicWeb
- ote_srv.FastInternet

ssl-policy

Server certificate location: http://certs.repository/server.cer

Server private key location: /usr/local/ssl/server.key

SSL version: TLS v1.1

Cipher suites

```
TLS_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
```

[✓] Client validation

Client certificate validation service: ote_srv.AccessControl

Allowed X509 client certificates

Certificates issued by DESEREC

Figure 10. Policy Console main view

The user is guided through the policy creation process by a creation wizard, which is specific to the type of policy being created. Wizards allow advancing in the creation process as the required information has been entered and assure a proper policy creation with the required parameters.

The console is implemented by following a plug-in based approach. That is, each supported policy type is implemented as a separate plug-in that is loaded by the console. This means that the set of supported policies can be extended as desired by developing additional plug-in modules.

This policy console can be used in the INTER-TRUST project to enable the definition of security policies and assign them to the different services that need to be managed. Although it is initially designed for the DESEREC project requirements and policies, it could be adapted to cope with the

INTER-TRUST project requirements including new policy types and parameters. The tool has been developed using Java and the policies are generated in an XML model. The plug-in mechanism makes it extensible and adaptable for new policy types. Thus, it could serve as basis for a policy editor tool supporting multiple kinds of policies.

4 Trust negotiation module: Xena

4.1 Overview

The security team of IT have developed a prototype named Xena which is defined to execute trust negotiation process. Xena framework is based on XACML (Extensible Access Control Markup Language) standard. Security policies of this framework are based on access control policies. Based on access control policies, we define attributes related to different elements of the policy (resources, subjects...). Xena is based on client/server architecture. A negotiation process can start between the two parties and it try to achieve an agreement to get access to a resource.

4.2 Xena architecture

Xena architecture is presented in Figure 11. It is a symmetric architecture, based on a classification of resources. Three classes are defined:

• Class 1 - “Resource with direct access”: All protected resources that belong to this class are managed by policies that do not trigger a negotiation process. If the needed attributes are given in the request, the evaluation of the respect of security policies may be possible. If they are not given, they will not be collected through a negotiation process.

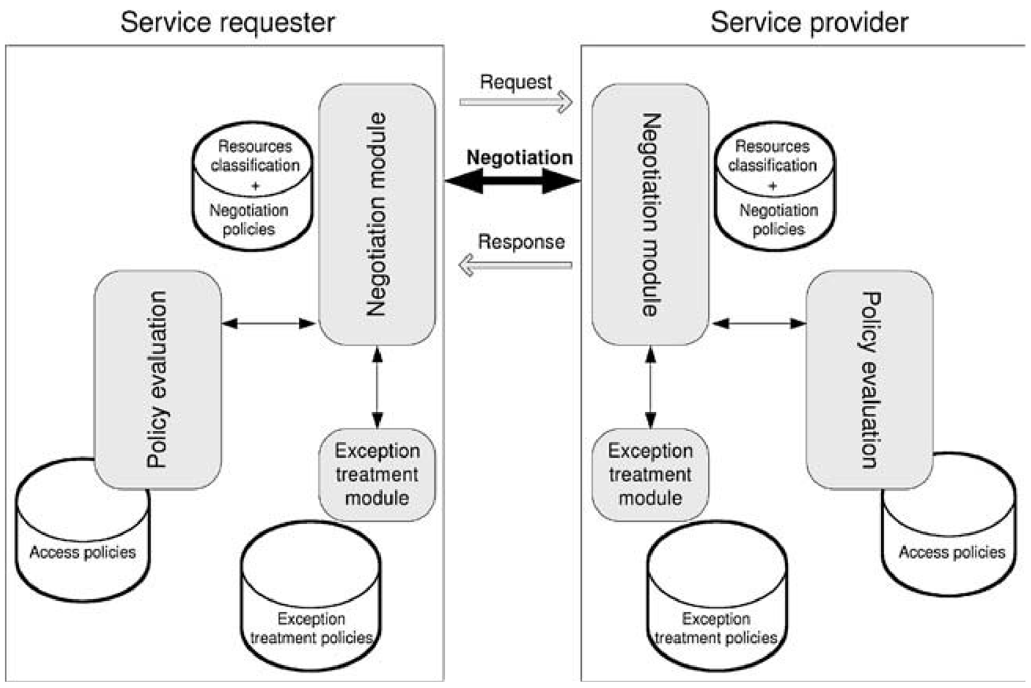

Service requester
Service provider
Request
Negotiation
Response
Resources classification + Negotiation policies
Negotiation module
Policy evaluation
Exception treatment module
Access policies
Exception treatment policies
Negotiation module
Resources classification + Negotiation policies
Policy evaluation
Exception treatment module
Access policies
Exception treatment policies

Figure 11. Negotiation architecture process

• Class 2 - “Resource with direct negotiated access”: If the request for access to resources from this class does not include all the needed attributes, the missing attributes may be directly requested from the requestor. “Directly requested” means that there is no strategy to hide the policy managing the resource’s access. This policy can be even revealed if the need arises.

• Class 3 -“Resource with indirect negotiated access”: We consider that the resources that are classified in this class are managed by a policy that should be kept secret. Thus, missing attributes may be indirectly requested from the requestor. That is, a strategy should be applied to obfuscate part (or all) of the negotiation policy.

Each entity (i.e. service provider or requestor) should have a classification of its own sensitive resources and its specific access control (i.e. interoperability) policies to protect these resources. It should also have negotiation policies. We have defined, for each entity involved in the process, a specific module that is in charge of the negotiation process. This negotiation module uses the negotiation policies to negotiate access to the protected resources. If an exception is raised, this module calls the exception treatment module so that it checks the exception treatment policies to find if there is a possible alternative.

4.3 Xena framework Implementation

Figure 12 represents the implementation of Xena prototype based on XACML standard. It shows where the negotiation module is introduced and different communications when a negotiation process is needed.

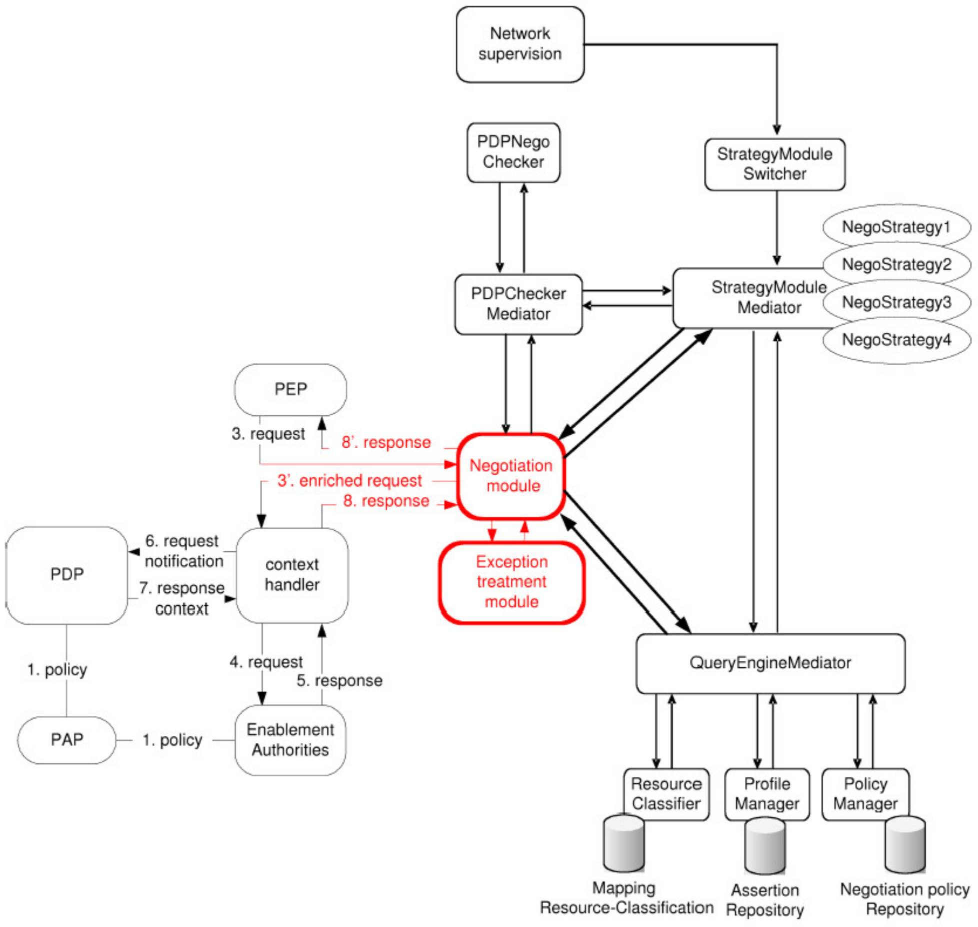


Figure 12. The implementation of the negotiation functionalities

The Negotiation Module is in charge of the core negotiation process. It has the following methods:

• The getResourceLevel method is used at the server side in order to get the classification of the resource. This method calls the processQuery method of the QueryEngineMediator that dispatches it to the ResourceClassifier module.

• negotiate method is first called at the server side whenever the resource's classification returned by the getResourceLevel is equal to 2 or 3. At the client side, this method is called whenever a notification (i.e. flag) is received from the server requesting a negotiation. "Negotiate" is the main trust negotiation method that takes an incoming message and generates a response. This method calls the StrategyModuleMediator, through the nextStep method, to decide about the negotiation strategy that applies to the current negotiation process. If an exception is detected, the negotiate method calls the ExceptionTreatmentModule that may propose alternatives.

5 Security policy interpreter

5.1 Overview

The INTER-TRUST Policy Interpreter is in charge of analysing and interpreting the security policies that have been previously defined by users and negotiated between entities. Thus, the Policy Interpreter should understand the conclusions inferred as the result of the policy reasoning process carried out by the policy engine. Additionally, the Policy Interpreter interfaces with the Aspect Generator module, for it to select and properly configure the set of appropriated aspects to be deployed.

5.2 Functional Specification

Figure 13 shows an overview of the Policy Interpreter within the INTER-TRUST architecture.

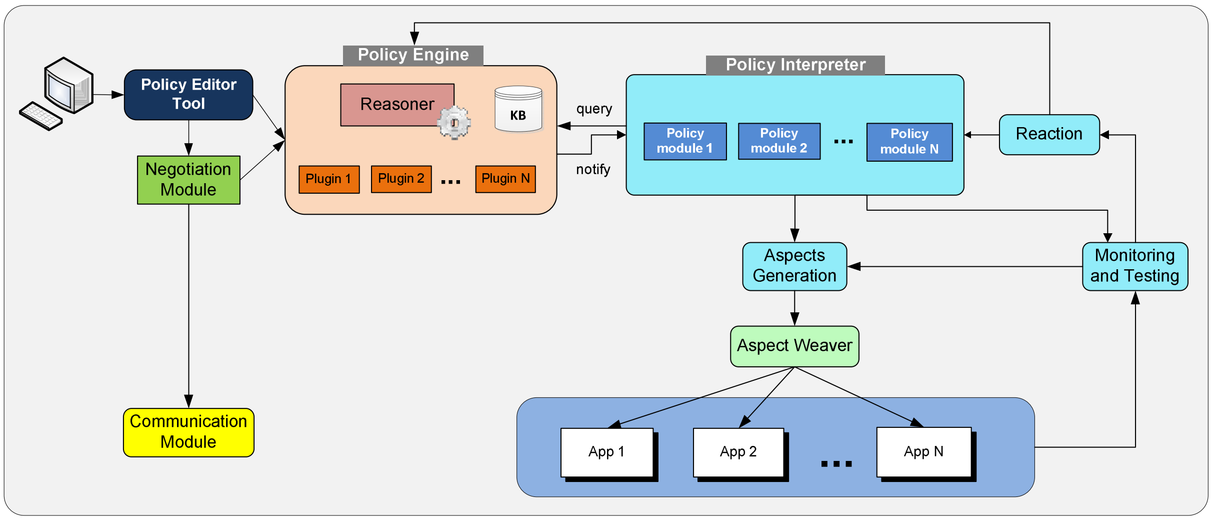


Figure 13. Policy Interpreter overview

As can be seen from the figure, the user first defines the policies using the Policy Editor tool (e.g. MotOrBAC). These policies are afterwards negotiated between the involved entities by means of the Negotiation module. The Communication module allows the communication between the entities to carry out such a negotiation. Then, the policies are evaluated in a Reasoner by the Policy Engine module. This process can infer new knowledge and make deductions over different aspects tackled in the policy. The inferred facts are stored in the Knowledge Base (KB) of the Policy Engine module with the aim of being later interpreted and handled by the Policy Interpreter module.

The Policy Engine and the Policy Interpreter are two different modules of the INTER-TRUST framework. Their interaction needs to be extensible to be able to cope with different kinds of policies and communication models. Moreover, the communication needs to be done seamlessly following either a push or pull mode. Thus, the Policy Engine has been designed following an extensible approach that uses plugins to increase its capabilities and to interact with the policy model kept in the Policy Engine. Likewise, this plugin mechanism allows an interaction with the Policy Interpreter that can, in turn, query the engine and access to the policy model information.

The Policy Interpreter is responsible for analysing the information obtained from the engine and making conclusions depending on the specific business logic implemented by the interpreter. For instance, the interpreter can allow users to access some resources under given circumstances, adopt a given privacy policy configuration, use a particular security communication channel, cypher or sign some particular messages in a given way, etc.

Like the Policy Engine module, the Policy Interpreter has been designed modularly. This approach allows different modules to manage different kinds of security policies inside the security Policy Interpreter. Usually, a given Policy Engine plugin interacts with a particular Policy Interpreter module in order to manage specific kind of security policies. But, it is not a restriction; different policy engine plugins can interact with one or more Policy Interpreter modules.

The Policy Interpreter analyses the deduced policy model and takes security decisions that can imply changing the deployment configuration. This reconfiguration/adaptability is performed by the Aspect Generation module that changes the aspects of the applications following an Aspect Oriented Programming (AOP) approach.

The Policy Interpreter is in charge of informing the Aspect Generator module for it to select the proper aspect(s) to be deployed in a given time. The Policy Interpreter should not be aware of which particular aspects are implemented by particular applications. To deal with this issue, a trade-off solution is to perform a best matching between the security policy requirements demanded by the Policy Interpreter and the capabilities offered by particular aspects. This approach allows the aspect generator to select the best aspect to be deployed. Thus, the Policy Interpreter provides the Aspect generator the information needed to correlate the different security policies of an application with the aspects that need to be dynamically incorporated to the application in order to satisfy each negotiated security policy.

The Monitoring and Testing module receives policy information from the Policy Interpreter and the execution traces coming from applications to be able to compare the behavior according to the security policy. This module invokes the Reaction module in case a reaction should be done. In turn, the reaction module communicates the context and reaction information to the Policy Engine and the Policy Interpreter for them to activate the new security policy that should be deployed.

Some aspects (or kinds of aspects) do not require additional information to work, but others may require as input a particular configuration coming from the interpreter. The Policy Interpreter may deliver a security policy configuration in a high level fashion following a common data format understandable by the Aspect Generator (or the module in charge of actually deploying and configuring the aspects).

5.3 Policy Interpreter usage example

As an example, let us consider a network filtering policy that is defined by the administrator to be applied to a firewall running in some component, e.g. a Central ITS Station Host. The policy enables access to the Central ITS-S host by the Road Side Units of the road operator.

After policy definition and negotiation, there is a policy representation in the knowledge base of the Policy Engine. This representation consists of a set of logical facts that represents an ontology model that could be expressed in RDF. In this example, we consider an ontology model of the policy based on the Common Information Model (CIM) standard defined by the DMTF[4]. The CIM Network Model includes a common definition of management information related to network filtering. Among others, it defines the FilterList class that represents a list of filter entries which compose a network filter condition. A FilterList is associated to the network device (e.g., a ComputerSystem) by the association HostedFilterList. In this example, we consider the HostFilterEntry to provide a high level definition for source or destination based on a referenced ComputerSystem instance.

The policy module of the Interpreter in charge of filtering policies interacts with the Policy Engine to get the filtering policy corresponding to the Central ITS-S Host. It performs a query in the ontology contained in the knowledge base through the corresponding plug-in module. Querying the ontology includes instance recognition (i.e., testing if an instance is belonging to a given class) and inheritance recognition (i.e., testing if a class is a subclass of another class and if a property is a subproperty of another). For example, a query using SPARQL language to obtain all filter lists associated to the Central ITS-S Host could be the following:

```
SELECT ? filterlist
WHERE { ? filterlist HostedFilterList #CentralITS-SHost };
```

SPARQL language is an SQL-like Semantic Web language for querying the system description and the firewall configuration using a given search criteria. It makes the complete management of such firewall policies, even having huge databases of policies, easier.

Making use of rules like the one above, the interpreter is able to get the information needed from the knowledge base of the Policy Engine. This will result in the specific filtering policy representation based on the ontology model. For instance, the following model represents the information obtained by the interpreter for the considered example:

```
<rdf:RDF>
  <ComputerSystem rdf:about="#CentralITS-SHost">
    <Name>centralhost</Name>
    <HostedFilterList rdf:resource="#CentralHostFilterList"/>
  </ComputerSystem>
  <ComputerSystem rdf:about="#RSU1">
    <Name>rsu1</Name>
  </ComputerSystem>
  ...
  <FilterList rdf:about="#CentralHostFilterList">
    <Name>Central host firewall filters</Name>
    <EntriesInFilterList rdf:resource="#RSUSources"/>
    <EntriesInFilterList rdf:resource="#..."/>
  </FilterList>
  <HostFilterEntry rdf:about="#RSUSources">
    <Name>Allowed central host sources</Name>
    <TypeOfEntry>Source</TypeOfEntry>
    <HostOfFilterEntry rdf:resource="#RSU1"/>
    <HostOfFilterEntry rdf:resource="#RSU2"/>
    ...
  </HostFilterEntry>
</rdf:RDF>
```

This information is then used by the interpreter to select the aspect to perform the policy enforcement. For instance, a filtering aspect may be chosen by the interpreter to apply this policy.

The above information will also serve for the policy Interpreter to generate the data that will be needed by the aspect to perform the actual enforcement. That is, the filtering policy to be applied.

[4] http://www.dmtf.org/

This data will be passed by the interpreter in a common format (e.g. XML) that will be understood by the aspect generator module. This information will be lately processed to generate the specific system rules (e.g. IP tables) depending on the specific system on which the enforcement is to be done. This generation may also take into account system specifics like obtaining the IP addresses for the corresponding referenced instances (e.g. rsu1).

6 Aspects generation module

6.1 Overview

This module receives notifications about a new security policy to be deployed, or about changes on the current security policies (as the result of the negotiation phase, to respond to the runtime monitorization of the system, etc.), and dynamically generates the list of aspects that have to be added/deleted from the application in order to conform to the new policy. The output of this module is the input to the *Aspect Weaver Module* described in the next section. Figure 14 provides a general overview of the interactions between the Aspect Generation Module and the rest of the modules in the INTER-TRUST framework architecture.

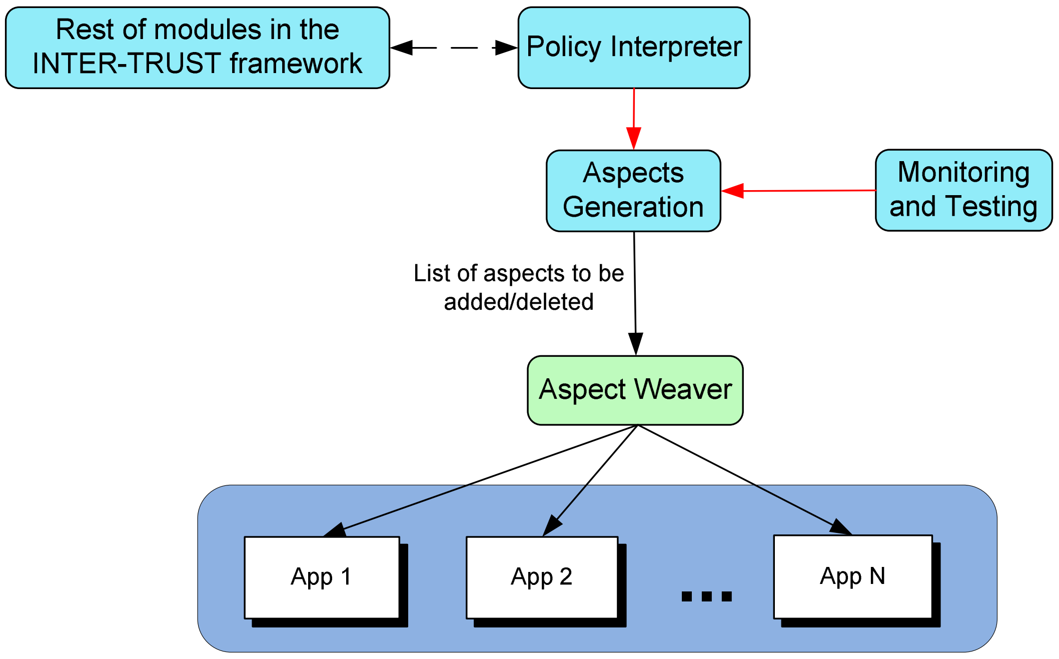


Figure 14. General overview of the interactions with the aspects generation and aspects weaver modules

The internal components that are part of the Aspect Weaver Module are specified in Figure 15. The details of these components are further specified in section 6.3, where the functionality of this module is detailed.

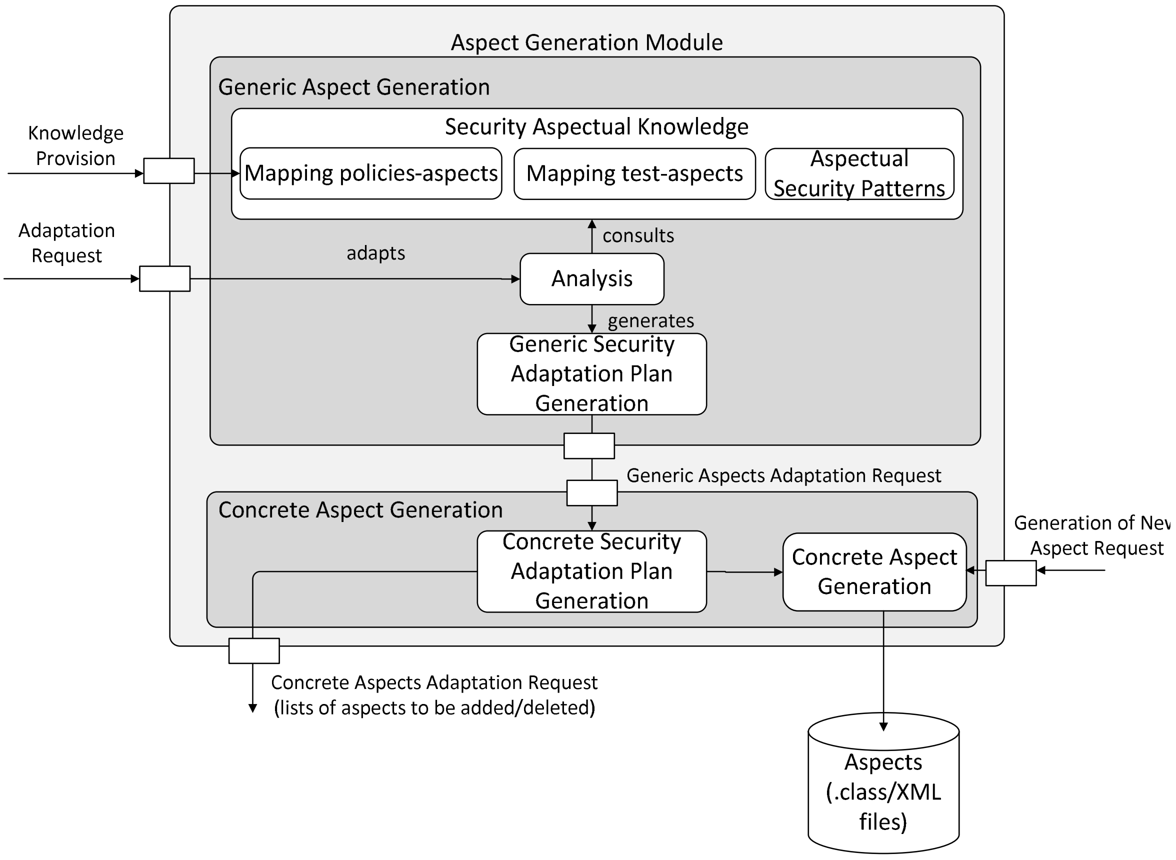

Aspect Generation Module
Generic Aspect Generation
Security Aspectual Knowledge
Knowledge Provision
Mapping policies-aspects
Mapping test-aspects
Aspectual Security Patterns
Adaptation Request
adapts
consults
Analysis
generates
Generic Security Adaptation Plan Generation
Generic Aspects Adaptation Request
Concrete Aspect Generation
Concrete Security Adaptation Plan Generation
Concrete Aspect Generation
Generation of Nev Aspect Request
Concrete Aspects Adaptation Request (lists of aspects to be added/deleted)
Aspects (.class/XML files)

Figure 15. Internal Structure of the Aspect Generation Module

6.2 Use Cases

The following use cases have been defined for the Aspect Generation Module:

- **Use Case 1**: When the framework is first instantiated, the application developer using the INTER-TRUST framework needs to provide information about the mapping between security policies and aspects and the mapping between the security tests and aspects. Moreover, a list of aspectual security patterns ready to be instantiated during the generation of aspects will be available in order to identify the join points where aspects need to be woven/unwoven. This information is available at runtime to generate the security adaptation plan.

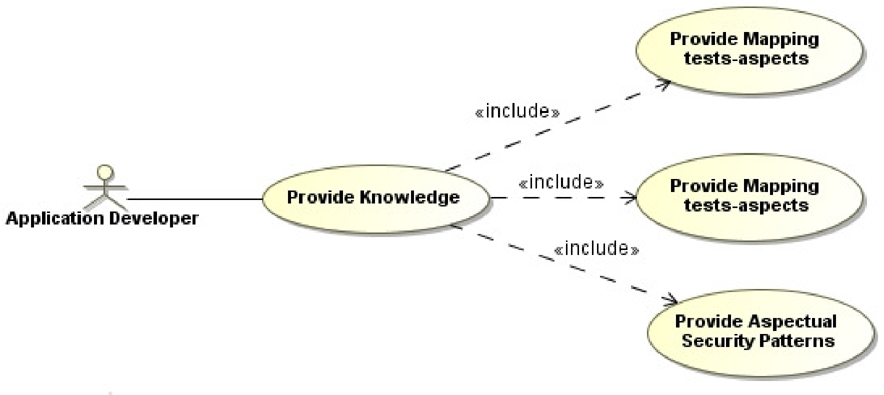

Application Developer
Provide Knowledge
«include»
«include»
«include»
Provide Mapping tests-aspects
Provide Mapping tests-aspects
Provide Aspectual Security Patterns

Figure 16. UML representation of use case 1

- **Use Case 2:** At runtime, the rest of modules in the framework could update the knowledge initially provided to this module.

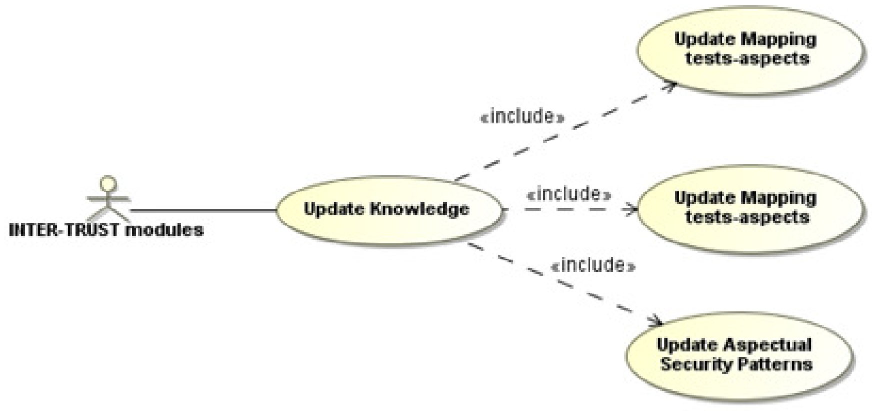

INTER-TRUST modules
Update Knowledge
«include»
Update Mapping tests-aspects
«include»
Update Mapping tests-aspects
«include»
Update Aspectual Security Patterns

Figure 17. UML representation of use case 2

- **Use Case 3:** The policy interpreter module will interact with the Aspect Generation Module to deploy the negotiated security policy.

- **Use Case 4:** At runtime, any change on the security policy needs to be notified to the Aspect Generation Model that will analyse the change from the aspect implementation point of view, and will decide which aspects need to be added/deleted in order to conform the new security policy.

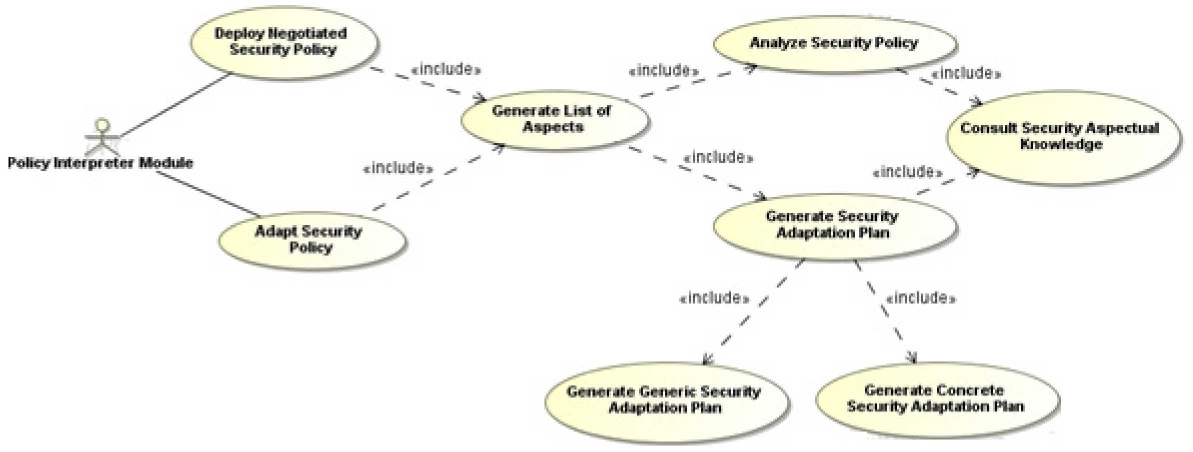

Deploy Negotiated Security Policy
Policy Interpreter Module
Adapt Security Policy
«include»
«include»
Generate List of Aspects
«include»
Analyze Security Policy
«include»
Consult Security Aspectual Knowledge
«include»
«include»
Generate Security Adaptation Plan
«include»
«include»
Generate Generic Security Adaptation Plan
Generate Concrete Security Adaptation Plan

Figure 18. UML representation of use cases 3 and 4

- **Use Case 5:** A new aspect or interception point (both pointcut[5] + advice[6] or only the pointcut) can be generated at runtime. This capacity of the Aspect Generation Module depends on the AOP weaver in use. For some weavers this possibility will not be available. The generation can be initiated by the Aspect Generation Module itself as part of the generation of the security adaptation plan or can be initiated by a different module in the INTER-TRUST framework.

[5] A pointcut is a program element that picks out join points and exposes data from the execution context of those join points. Pointcuts are used primarily by advice. They can be composed with boolean operators to build up other pointcuts. More details are provided in D4.1 deliverable.

[6] A piece of advice brings together a pointcut and a body of code to define aspect implementation that runs at join points picked out by the pointcut. More details are provided in D4.1 deliverable.

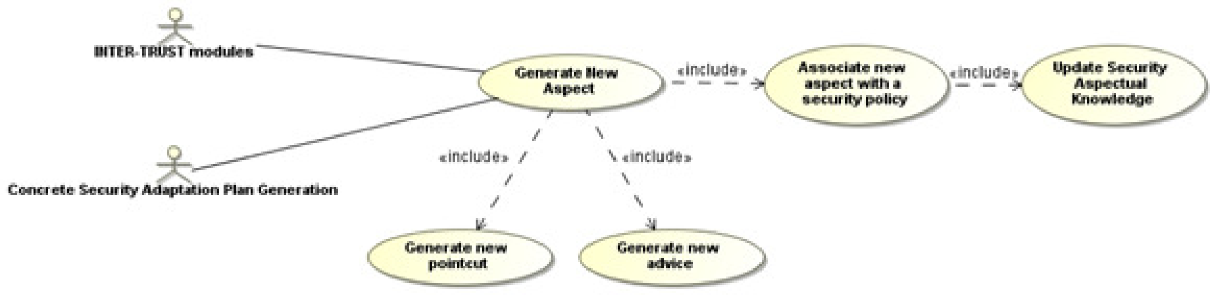

INTER-TRUST modules
Concrete Security Adaptation Plan Generation
Generate New Aspect
«include»
Associate new aspect with a security policy
«include»
Update Security Aspectual Knowledge
«include»
«include»
Generate new pointcut
Generate new advice

Figure 19. UML representation of use case 5

6.3 Functional Specification

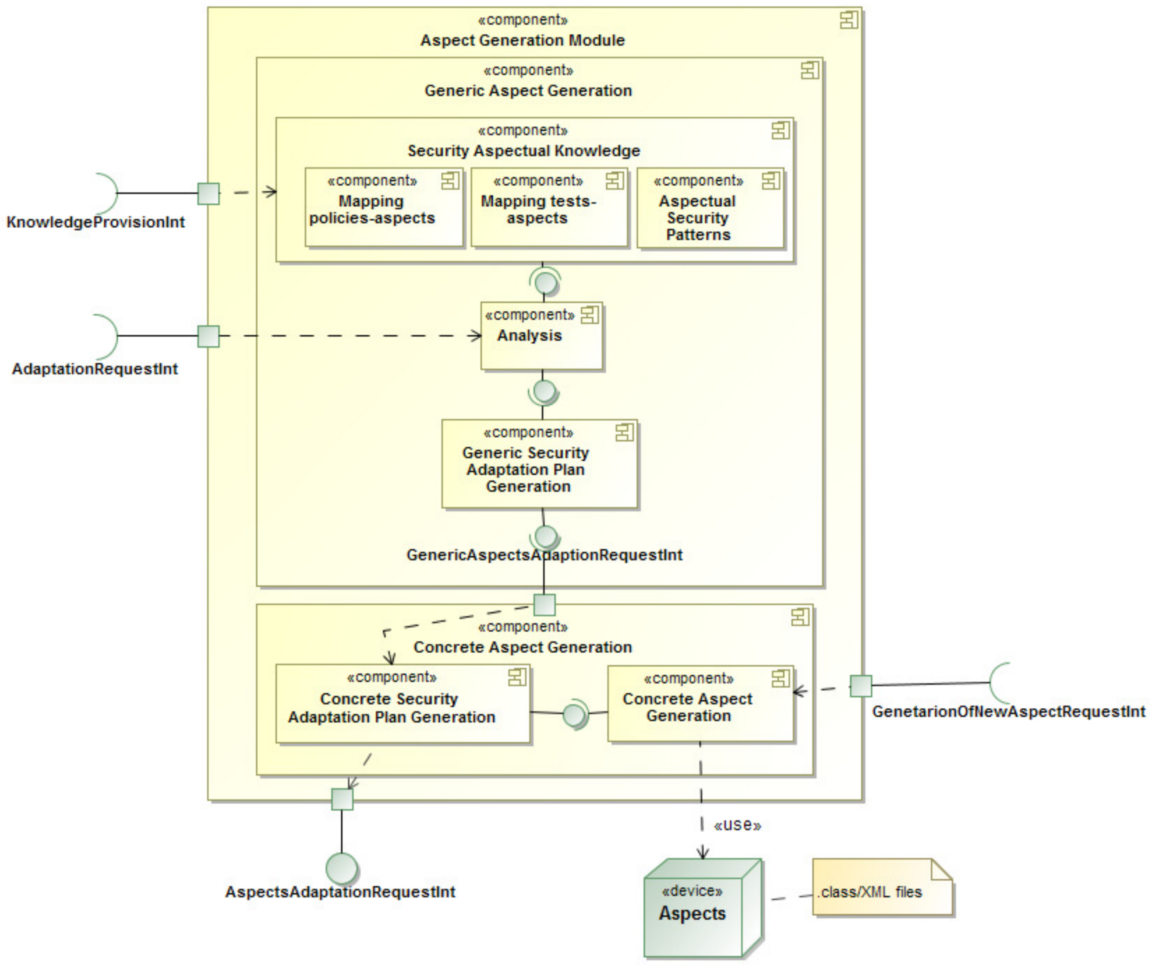

«component»
Aspect Generation Module
«component»
Generic Aspect Generation
«component»
Security Aspectual Knowledge
«component»
Mapping policies-aspects
«component»
Mapping tests-aspects
«component»
Aspectual Security Patterns
KnowledgeProvisionInt
«component»
Analysis
AdaptationRequestInt
«component»
Generic Security Adaptation Plan Generation
GenericAspectsAdaptionRequestInt
«component»
Concrete Aspect Generation
«component»
Concrete Security Adaptation Plan Generation
«component»
Concrete Aspect Generation
GenetarionOfNewAspectRequestInt
AspectsAdaptationRequestInt
«use»
«device»
Aspects
.class/XML files

Figure 20. UML Component diagram of the Aspect Generation Module

Figure 20 shows a UML composite component with the internal components of the Aspect Generation Module. Since the generation of aspects partially depends on the AOP framework being used by an instance of the INTER-TRUST framework, this module is organised in two main sub-modules: (1) the *Generic Aspect Generation* sub-module, which generates the part of the generic security adaptation plan that is generic and independent from a particular AOP framework, and (2) the *Concrete Aspect Generation* sub-module, which generates the part of the generic security adaptation plan that is concrete for a particular AOP framework. This second sub-module also generates at runtime new aspects for a concrete AOP framework (only pointcuts and/or pointcuts+advices according to the functionality offered by each AOP framework). In the rest of this section we describe these sub-modules with more details.

Generic Aspect Generation sub-module. This module is in charge of deciding which aspects need to be added/deleted to an application in order to deploy a new security policy or to adapt a previously deployed security policy. Its input are: (1) the information needed to generate the security adaptation plan (`KnowledgeProvisionInt` interface in **Errore. L'origine riferimento non è stata trovata.**), (2) requests to adapt the security policy or the security tests (`AdaptationRequestInt` interface in **Errore. L'origine riferimento non è stata trovata.**), and (3) requests to add new aspects into the aspect repository (`GenerationNewAspectRequestInt` interface in **Errore. L'origine riferimento non è stata trovata.**). Its output is a generic security adaptation plan with the list of aspects that need to be added/deleted, without including any implementation details of the AOP framework being used (`GenericAspectsAdaptationRequestInt` interface in **Errore. L'origine riferimento non è stata trovata.**). These implementation details will be later added by the Concrete Aspect Generation sub-module in order to complete the security adaptation plan. It is composed by three internal components: Security Aspectual Knowledge, Analysis and Generic Security Adaptation Plan Generation.

- **Security Aspectual Knowledge**. This component represents all the information that this module needs in order to adapt the applications to changes on the security policies. At least, the following information is required: (1) a mapping table with the information needed to relate the different security policies of an application with the aspects that need to be dynamically incorporated to the application in order to satisfy each negotiated security policy, (2) a mapping table with the information needed to relate the different security tests that can be performed with the aspects that need to be dynamically incorporated to the application in order to perform the tests, and (3) a list of aspectual security patterns (i.e. the security-related components and connections that must be added/deleted/modified in order to fulfil a given security policy). When needed, these components and connections will be parameterized in order to specify the list of possible joinpoints[7] in terms of the application components and connections. This information is part of the input to this module, it is mainly defined at design time; it must be specified independently from the AOP framework used to weave the aspects, and it depends on the applications using the INTER-TRUST framework.

- **Analysis.** The *analysis* component in the *generic aspect generation module* analyses the notified changes and the security aspectual knowledge and determines whether the security aspects, currently instantiated in the application, fulfil the new security policy or some changes must be done in the current aspects.

- **Generic Security Adaptation Plan Generation.** This component generates a list of aspects and/or pointcuts that need to be added/deleted in order to satisfy the new security policy. This component is independent from the AOP framework used to weave the aspects. The output of this component is the input to the *concrete aspect generation* sub-module.

[7] A joinpoint is a well-defined point in the execution of a program. More details are provided in D4.1 deliverable.

Concrete Aspect Generation sub-module. This module completes the generic security adaptation plan generated by the *generic aspect generation sub-module* with information specific of the AOP framework used for a particular instantiation of the INTER-TRUST framework. Its input is the generic security adaptation plan (`GenericAspectsAdaptationRequestInt` interface in **Errore. L'origine riferimento non è stata trovata.**), and its output is a concrete security adaptation plan (`Concrete AspectsAdaptationRequestInt` interface in Figure 20). It is composed by two internal components: Concrete Security Adaptation Plan Generation and Concrete Aspect Generation.

- **Concrete Security Adaptation Plan Generation.** As previously indicated, in this component the generic security adaptation plan generated by the *generic aspect generation sub-module* is concretized with information that is specific of each AOP framework.
- **Concrete Aspect Generation.** This component generates the weaving information in case new aspects or pointcuts must be incorporated to the application. For example, in AOP JBoss, this component may generate an XML file with the aspects and pointcuts that must be instantiated by the aspect weaver.

Note, that these components are completely dependent from the AOP framework used in a particular instantiation of the INTER-TRUST framework. We intentionally have included them in an internal sub-module of the *aspect generation module* in order to make this dependency transparent to the rest of the components in the INTER-TRUST framework.

6.4 Interfaces Specification

The Aspect Generation module has three input interfaces and one output interface.

KnowledgeProvisionInt. This input interface is used by the rest of modules in the INTER-TRUST framework architecture to provide/update the application dependent knowledge.

AdaptationRequestInt. This input interface is used by the rest of modules in the INTER-TRUST framework to request an adaptation in the aspects being applied for a particular application. This information is provided in terms of changing that need to be done on the security policies or on the security testing.

GenerationOfNewAspectRequestInt. This input interface is used by the rest of the modules in the INTER-TRUST framework to request the incorporation/generation at runtime of a new aspect (this can be only a new pointcut or both a poincut and an advice depending on the AOP weaver being used in each instance of the framework) into the INTER-TRUST framework.

AspectsAdaptationRequestInt. This output interface provides information about the list of aspects that need to be added/deleted by the aspect weaver.

6.5 Unit Tests Specification

The following tests have been identified:

Test 1. The correct initialization of the knowledge required by the Aspect Generation Module will be tested.

Test 2. The completeness of the knowledge required by the Aspect Generation Module will be tested to identify if new knowledge is required.

Test 3. The correct update at runtime of the knowledge required by the Aspect Generation Module will be tested.

Test 4. The correct deployment of a security policy by the addition/deletion of aspects by the Aspect Generation Module will be tested.

Test 5. The correct generation of new aspects (only new pointcuts and/or new pointcuts+advice) will be tested when allowed by the AOP weaver.

7 Aspect weaver module

7.1 Overview

The INTER-TRUST aspect weaver module will allow adding and deleting aspects. As shown in Figure 21 this module executes the *adaptation plan* generated by the aspect generation module (the list of aspects to be added/deleted). Thus the information generated by the Aspect Generation Module is the input to the *Execute the Security Adaptation Plan* component which will be specifically implemented for each AOP framework that may be used in the project. This component is a wrapper that will translate the list of aspects received as input (which is specified independently of a particular AOP framework) to the particular syntax of the particular AOP weaver being used. This means that different instantiations of the INTER-TRUST framework for using different AOP weavers will provide different implementations of this component. The output of this component is a direct interaction with the selected AOP weaver in order to interact with it and to weave/unweave the corresponding aspects into the applications.

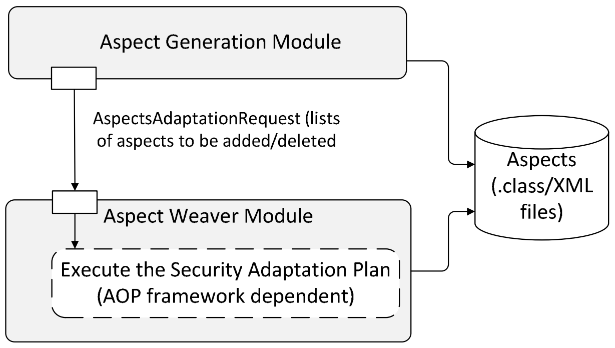

Aspect Generation Module
AspectsAdaptationRequest (lists of aspects to be added/deleted
Aspects (.class/XML files)
Aspect Weaver Module
Execute the Security Adaptation Plan (AOP framework dependent)

Figure 21. Internal Structure of the Aspect Weaver Module

7.2 Use Cases

The following use case has been defined for the Aspect Weaver Module:

- **Use Case 1**: The Aspect Weaver module receives a list of aspects that need to be added/deleted, translates this information to a format that can be understood by the particular AOP weaver being used in the instance of the INTER-TRUST framework and executes the corresponding actions.

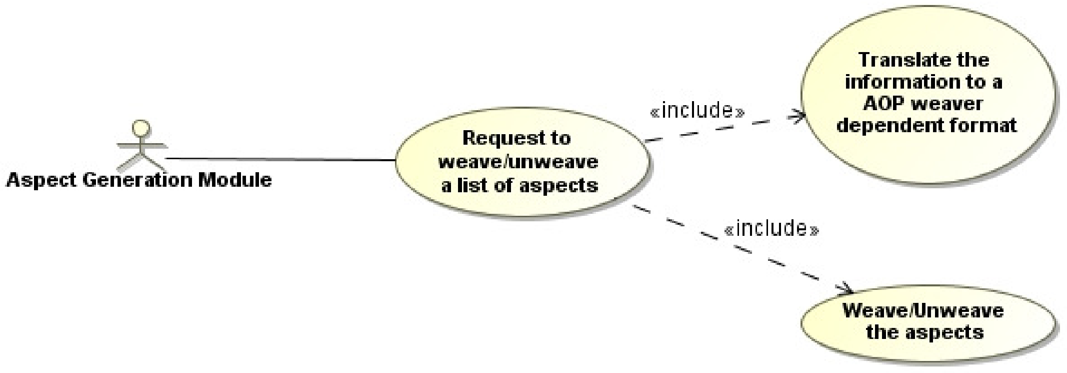

Aspect Generation Module
Request to weave/unweave a list of aspects
«include»
Translate the information to a AOP weaver dependent format
«include»
Weave/Unweave the aspects

Figure 22. UML representation of use case 1

7.3 Functional Specification

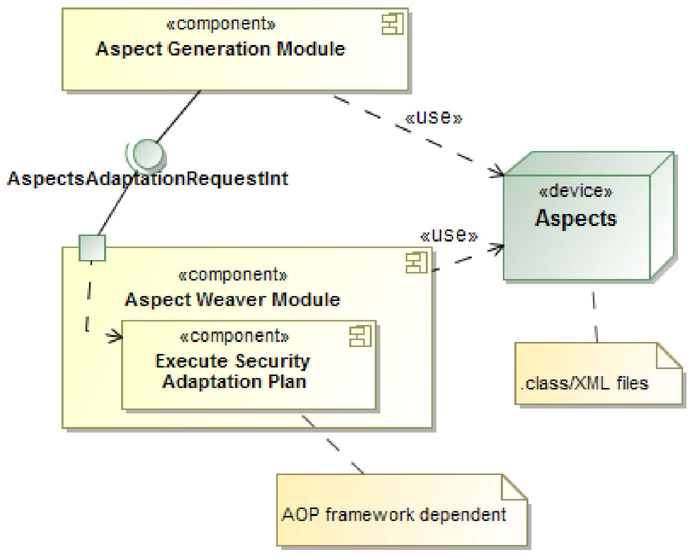


Figure 23. UML Component diagram of the Aspect Weaver Module

Figure 23 shows a UML composite component with the internal component of the Aspect Weaver Module and the interaction with the Aspect Generation Weaver through the interface specified in the next section.

7.4 Interfaces Specification

This module has one input interface. No explicit output interface is represented because, as previously said, this component is a wrapper of the particular AOP weaver being used in each instance of the framework.

AspectsAdaptationRequestInt. This input interface is used to receive the list of aspects that have to be woven/unwoven.

7.5 Unit Tests Specification

The following tests will be performed and repeated for each “platform X” used by an instance of the INTER-TRUST framework:

Test 1 for platform X. The correctness of the translation from the input to this module (list of aspects to be added/deleted) to the particular actions required by each AOP weaver in order to weave and/or unweave an aspect.

Test 2 for platform X: As a result of weaving/unweaving a set of aspects, the changes on the security policy must be exactly the changes that were requested by the INTER-TRUST framework.

8 Monitoring aspect module

8.1 Overview

The main objective of this module is to continuously capture observable information at different levels (e.g., physical environment, hardware, network, operating system, end-user specific applications etc.) and to send them to the MMT monitoring tool for security analysis (more details about MMT monitoring tool are provided in section 11.1). This information is also used at the application level in order to be able to make decisions according to the deployed security policies with contextual constraints.

The monitoring module is an aspect weaved into the application code via pointcuts in order to take security-related decisions or notify the event to MMT for analysis. It is composed of a set of functions (i.e., Java methods) that can be weaved when necessary.

Based on the e-voting requirements defined in D2.1.1, some devices have limited processing power and consequently cryptographic operations can be slow on those devices. Delegation of processing is a solution to delegate costly computation to more powerful devices. Thus, device computation power needs to be measured by the monitoring aspect module, if the necessary access rights are available. Notice that this indicator can change during run time since it depends on the number of running applications on the device and on their CPU consumption.

8.2 Use cases

The monitoring aspect module should be able to measure several indicators at different levels (e.g., CPU usage value at the system level). Several functions within this aspect are generic and can be used in different contexts (mainly those related to the physical environment, hardware and operation system) and others are application or device specific. If enough access rights are available during the application operation, the monitoring aspect module is able to send application internal messages to the Montimage Monitoring Tool (MMT) for analysis. More details about MMT are provided in section 11.1.

According to these requirements, two use cases are defined for the monitoring aspect module:
- This module should be able to measure several metrics for contextual security policies. These metrics are application specific and can influence the deployed security mechanisms.
- It should also be able to notify MMT tool by sending application internal events (like what java logger would do).

8.3 Functional specification

The monitoring aspect module is composed of three complementary, but independent, modules as shown in Figure 24.

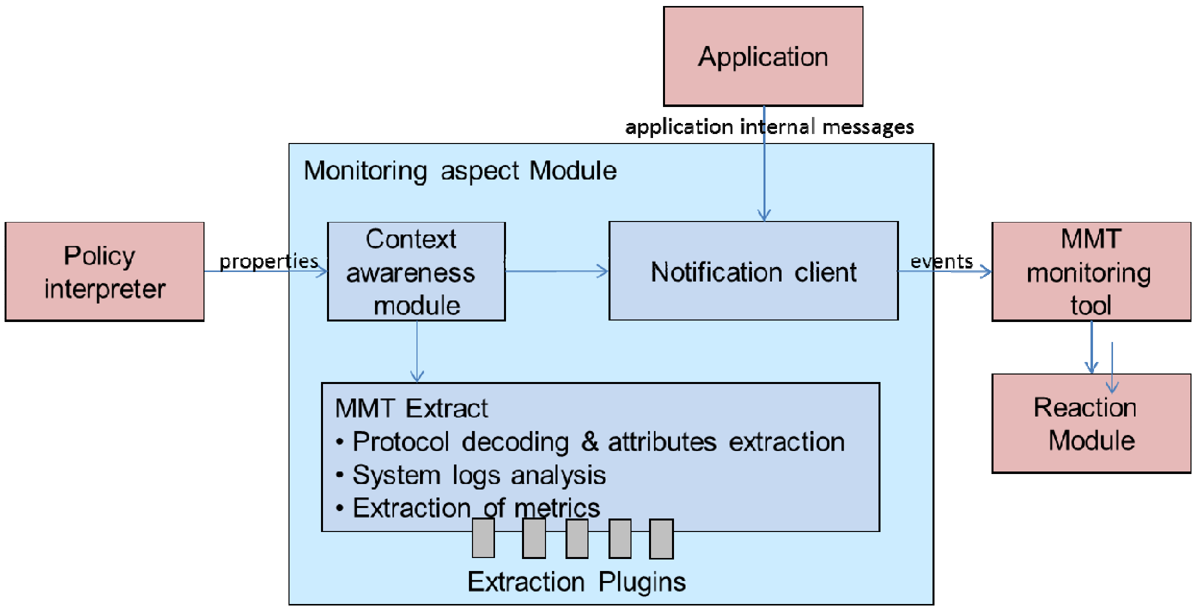


Figure 24. Conceptual view of the monitoring aspect module

- Context awareness module receives the negotiated security policy to be weaved in the system and identifies contextual security rules. The evaluation of this context will be the main task of this module composed of a set of librairies that allow the capturing of relevant data for their analysis based on MMT-Extract module.
- MMT-Extract is the core event processing module that permits to extract data attributes values from any structured data. MMT-Extract incorporates a plugin architecture for the addition of new data structures, logs, protocols parsing etc. and a public API for integrating third party probes. This component already exists in the current version of MMT. New plugins will be added to be able to manage the different INTER-TRUST case study scenarios.
- The notification client module will send relevant events from different levels (system, application or network) to MMT for their security analysis. These events are captured by the context awareness module, notified by the running application or captured from the network traffic.

8.4 Interfaces specification

The first input of the monitoring module is the security policy notified by the policy interpreter module that is to be deployed in the application after the negotiation phase. The XML format of these security policies are to be defined in a second step. This input is also an output of the monitoring module since it will inform the MMT standalone tool with the deployed security policies to be verified “online” and respected by the communication with interoperating systems.

The monitoring module will also capture (using the AOP framework) the application internal messages and events and send them (via the network) to a local or remote MMT server. It should produce HTTP based messages in the standard format of "field: value"[8] (i.e., POST messages for sending events and GET messages pour status commands).

[8] RFC2822.

8.5 Unit tests specification

Test 1: Different type of contextual security rules will be deployed and the monitoring module should be able to collect the environmental metrics relying on its internal libraries and evaluate the validity of the different contexts

Test 2: Some pointcuts are introduced in the application to notify relevant internal messages that are transferred by the monitoring module to MMT for performing a deep security analysis.

9 Injection testing module

9.1 Overview

The main function of Monitoring and testing modules is to stimulate the system in two different ways. During the development and testing phases, the adapted **Flinder fuzz testing tool** (SL) injects test data (via the Inject aspect) to perform active and fuzz testing on the target system (Target of Evaluation - ToE). During the development, testing, and operation phases, the tool captures application events to generate traces (via the Notification aspect) that can be used by the Monitoring tool (MI). During the testing and operation phases, monitoring techniques will be used to detect non-compliance with security requirements as well as other contextual information.

Flinder was originally built around the same concept as the INTER-TRUST project is aiming at: in a plug-in-able architecture it checks for certain well-defined types of flaws which may lead to security vulnerabilities. This is supported by Flinder both in black-box- and white-box-mode:
In black-box-mode Flinder works in a man-in-the-middle manner to modify (fuzz) inputs to create states that can lead to software failures. The core part of the architecture is composed of descriptor-configurable protocol and test logics, integrated with input capturers and output dispatchers to allow running tests on a plethora of computing platforms ranging from server products through personal computers to embedded systems. This is completed with plug-ins, describing and implementing test cases for certain types of programming bugs.
In the white-box-mode Flinder applies source-code-based testing. The architecture is similar to that in the black-box-mode with the difference that fuzzing is complemented by fault injection supported by code instrumentation. Again, the protocol logic and the test logic are the core parts, and plug-ins describe the specific vulnerabilities that are being looked for.

9.1.1 Basic operation of Flinder

Figure 25, below shows the main modules of the current version of Flinder, and the connections between them. This architecture shall be adapted to the INTER-TRUST Framework.

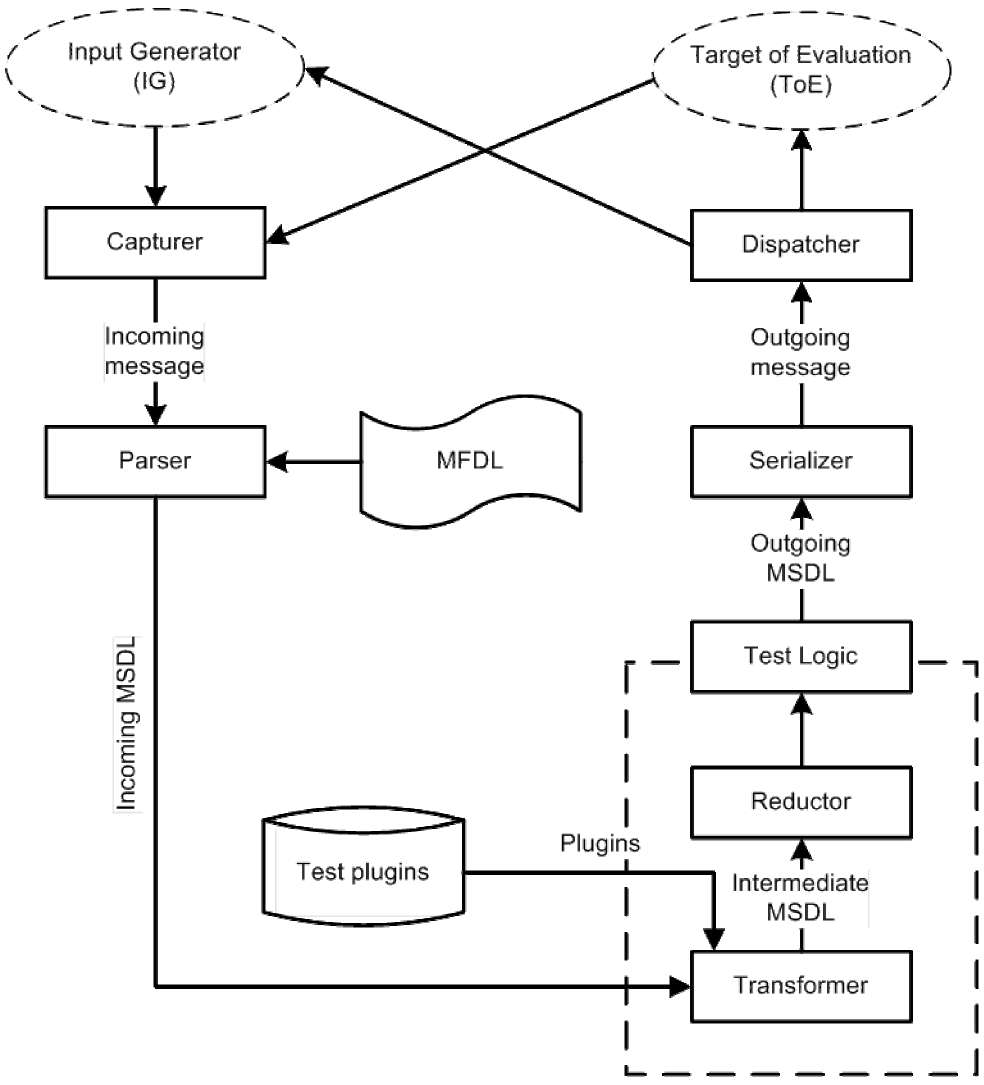


Figure 25. Basic operation of Flinder

Following we present a brief summary of the Flinder modules and their roles in test runs:

1. A valid file or protocol message sent by the **Input Generator,** or an execution result or protocol reply sent by the **Target of Evaluation** (e.g. the executable of the application under test) is intercepted by the **Capturer,** which transforms it into a Binary Message Envelope (BIME).
2. The **Parser** transforms the BIME into an XML-based message in the Flinder Message Structure Description Language (**MSDL**) based on a generic description of the incoming message. This description is expressed in the Flinder Message Format Description Language (**MFDL**) and is also an XML-based file.
3. The **Test Logic** module modifies certain fields in the MSDL message according to its transformation rules. These modifications are to uncover security-relevant programming bugs (such as integer overflows, buffer overflows or other typical errors). Consisting of a **Transformer, Reductor** and **Test Logic** parts, in order to be able to use test algorithms.

The **Transformer** generates *Intermediate MSDLs* based on fields to be modified present in the MSDL. We will have separate *Intermediate MSDLs* for all modified fields.

The **Reductor** receives all the given MSDLs from the transformer and generates the real test vector MSDLs based on all the existing listed values in the relevant *Test plug-ins* (for example the „integer boundary testing" test plug-in tests the values for: 0,1, MIN INT and MAX INT).
So the **Reductor** receives one *intermediate MSDL* (called from the transformer for each generate MSDL), and outputs several *outgoing MSDLs*.

4. After the MSDL message has been modified, the **Serializer** transforms it back into a BIME.
5. Finally, a **Dispatcher** delivers the modified raw message to the ToE or the IG, depending on the direction of the actual message. The Dispatcher can be generic, but it is typically tailored to the ToE and the IG.
6. Test case verdicts (the actual test results) are generated by the external **Actuator** module.

The core of the testing activity is the **generation of test vectors**, which is accomplished by algorithmic modification of MSDL messages in Flinder. In the original design, the Test Logic module of Flinder uses statically linked Test Suites, which generate test vectors in a pre-defined manner.

To have more detailed description of the specification of Flinder please see section 11.3.

9.2 Use case

Point-cuts shall be defined so that Flinder could perform the fault injection process into INTER-TRUST capable systems. In this context, fault injection refers to the injection of test vectors into the ToE through the fault injection testing module. Fault injection will be targeted at all interfaces of the use case scenarios, including interfaces responsible for the basic functionality of the use case as well as interfaces that will be created to implement the add-on functionality required by the INTER-TRUST framework.
In this case, the implementation of the Negotiation and Communication modules will be tested depending on the context, for all use case scenarios.

9.3 Functional specification

Requirement:
All subsystems (including applications, communication modules) using the INTER-TRUST Framework will be required to include/build in the point-cuts for the Notification and Injection aspects.

In the following section the architecture of the first adaptation of Flinder to INTER-TRUST Framework will be described and illustrated.

In Figure 26 below, the basic operation of Flinder before adaptation to INTER-TRUST Framework is illustrated.

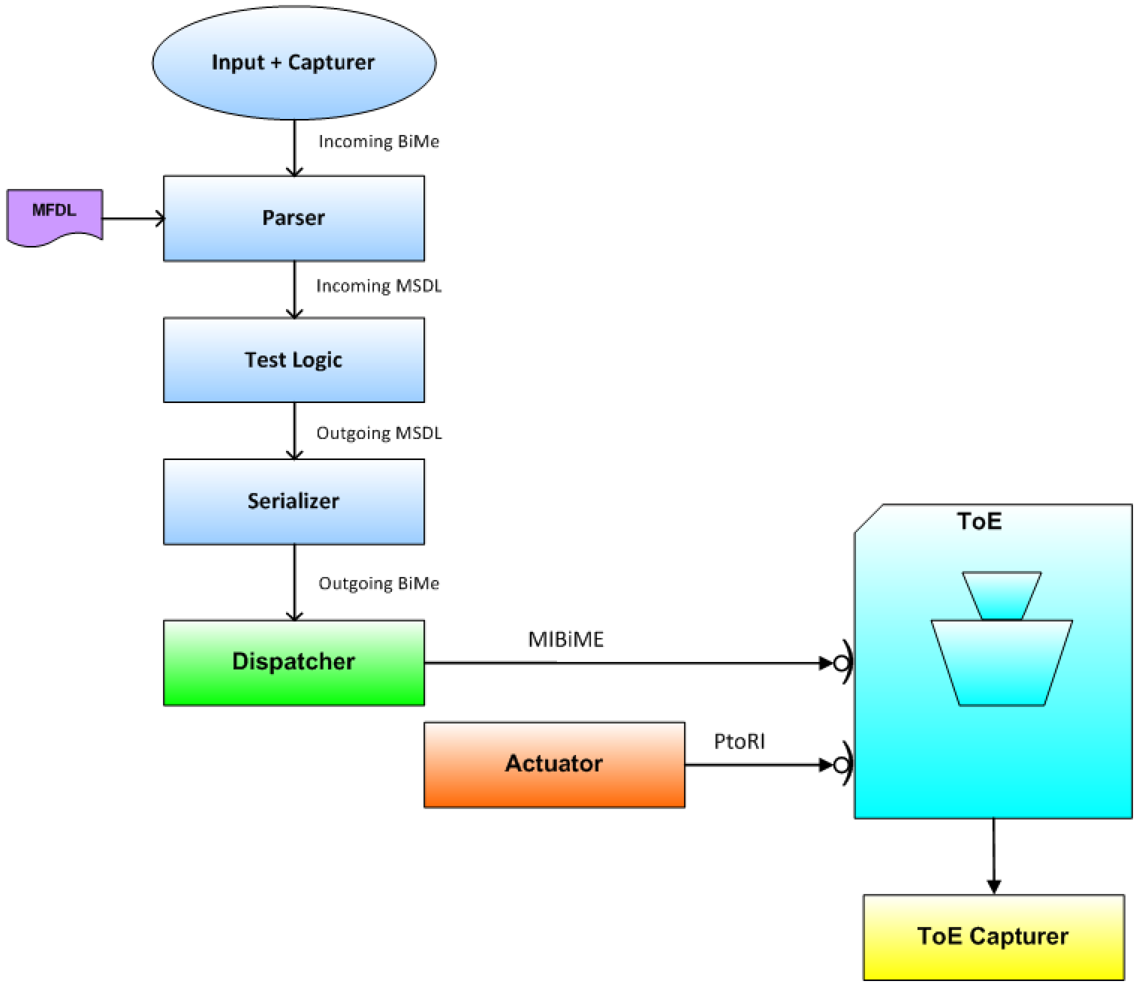


Figure 26. Basic operation of Flinder before adaptation to INTER-TRUST Framework

The fuzz-testing tool can currently only work on Targets-of-Evaluation (ToE) where the Input Capturer and output Dispatcher are implemented for the target system to be evaluated, or an earlier developed capturer-dispatcher pair can be reused. This is because Flinder makes no assumption on the ToE functionality and interfaces – this is one of the strengths of the Flinder architecture.
In the INTER-TRUST project, however, we are going to develop a framework, in which interfaces can be defined also for the sake of testability. We are going to define interfaces and redesign the Flinder architectural model to make it possible to easily perform standard vulnerability fuzz testing on any target system implementing the INTER-TRUST framework.

In Figure 27 below, the redefined Flinder architectural model is illustrated. This redesign assures the adaptation of Flinder to the INTER-TRUST framework.

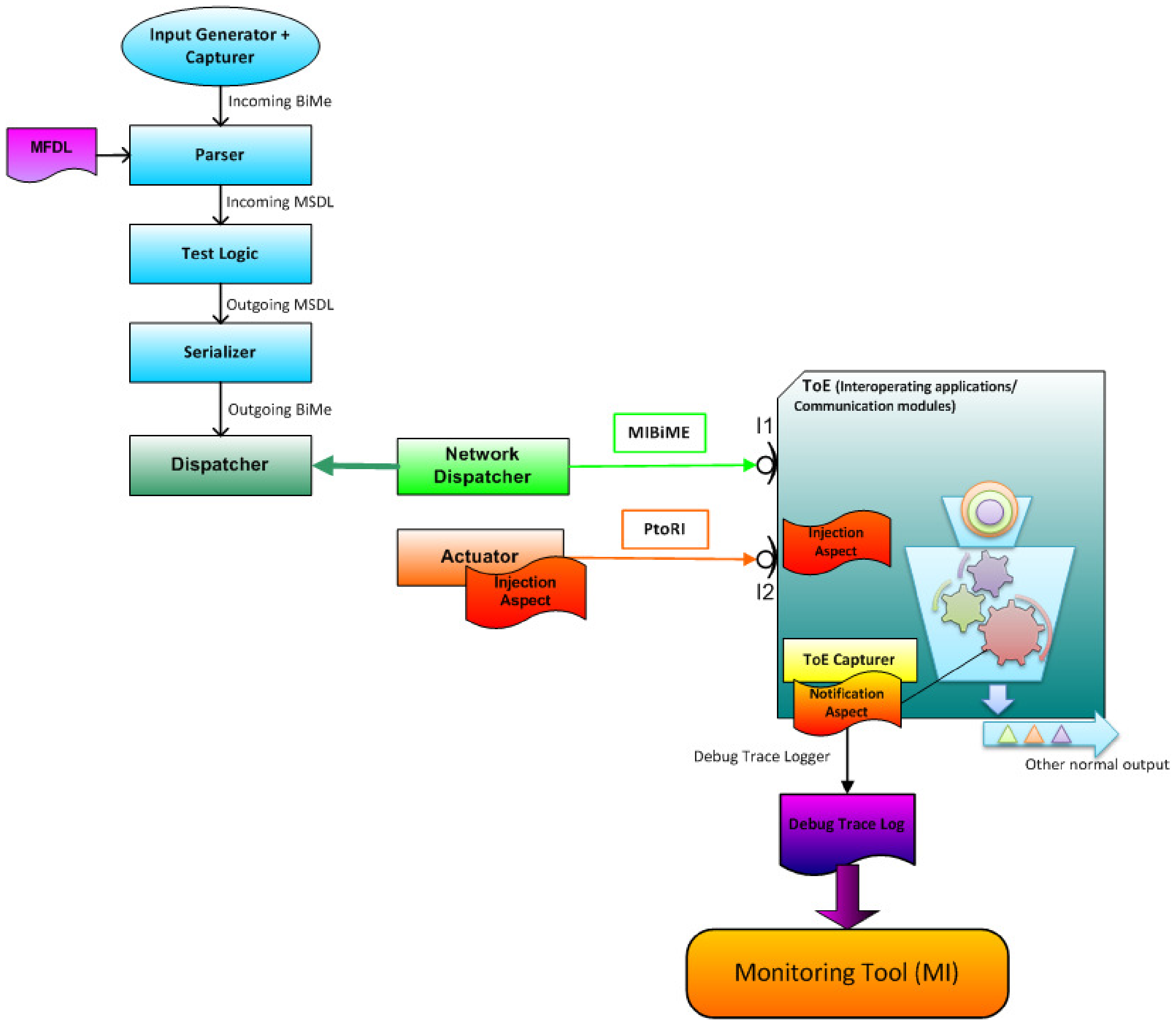


Figure 27. Flinder adaptation to INTER-TRUST Framework

BiME: Binary Message Exchange
MSDL: Message Structure Definition Languge
MIBiMe: Malicious Input generated by Flinder
PtoRI: Prepare to Receive Input
I1: ToE interface for any existing functionality of the ToE
I2: a new ToE interface, for injection (fuzz testing)

MFDL:
In order to adapt the current Flinder architecture and dynamic evaluation process to INTER-TRUST, a new Message Format Definition Languge shall be produced/written as an input for the system based on policy file formats. *(Comment: This can be done after the policy file format will be specified so that we will be able to have new MFDLs when the Policy Negotiating module is specified).*

Parser:
It generates an XML-based inner structure from the input that can be processed by Flinder in the later steps. (In order to parse the input, the MFDL has to be prepared based on the communication format- and on the policy file format).

Test Logic:
It modifies certain fields in the input message according to the parameters specified in the MFDL (e.g. indicating that a certain field is a string type) and Flinder's internal test generation rules (e.g. successive approximation of buffer size).

In Figure 28 below, there is a detailed illustration of the redesigned elements of Flinder:

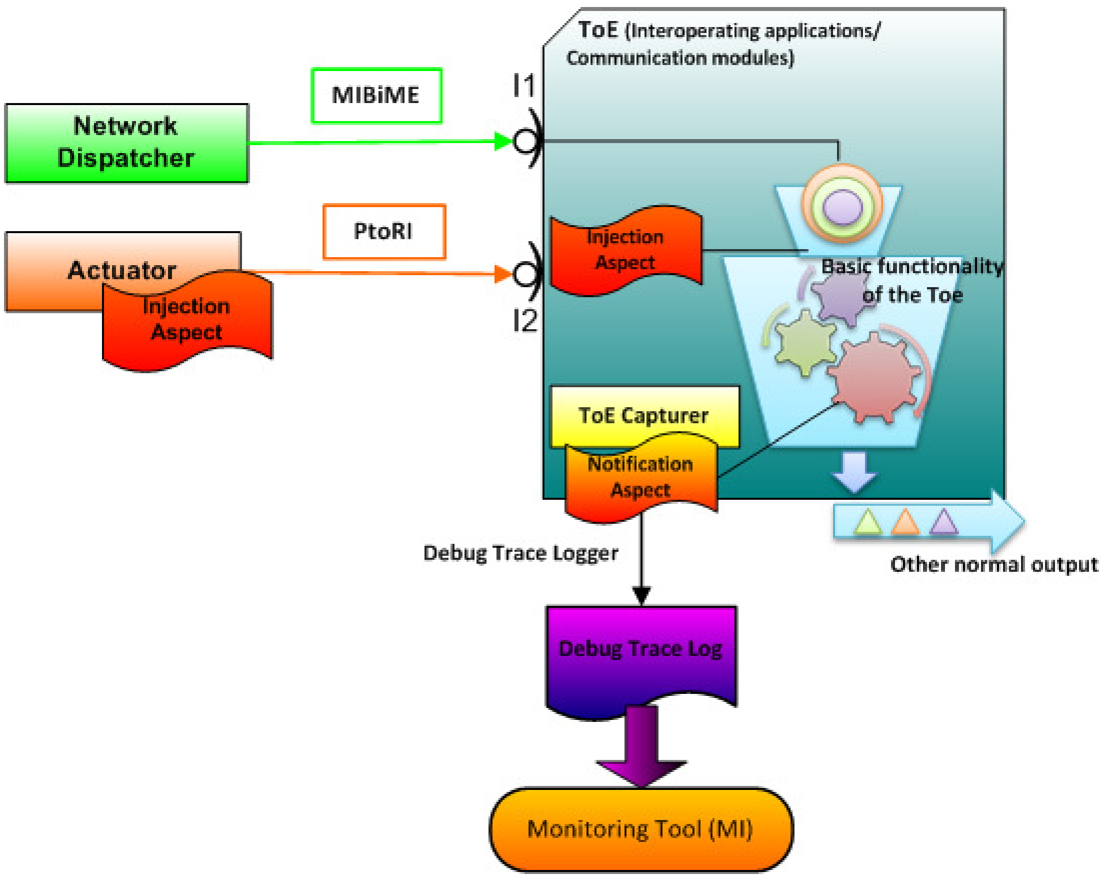


Figure 28. Redesigned elements of Flinder

ToE: Target of Evaluation including interoperating applications and communication modules.

Network Dispatcher:
The Network dispatcher waits for a time when the ToE interface is expecting input, and sends a Flinder-modified BiMe to the ToE.

Toe Capturer:
Capture application events to generate traces and internal state-> this would be weaved as the Notification aspect (test report generator)

Injection aspect:
The **Injection aspect (updated Actuator)** is responsible for making it possible to directly provide malicious input (test vectors) from the test modules into the communicating modules (Interoperating applications or Communication modules).
Injection aspect will decide whether the ToE will accept the Flinder-modified input. If the ToE is not ready to accept the input, the Injection aspect will manipulate the ToE to be in a state where it is ready to receive input (e.g. through a reset)

Notification aspect:
The **Notification aspect** will be responsible for making sure that the inner state of the different communication modules is monitorable by the Monitoring Modules. The monitoring module will also detect if the processing of a particular test vector caused an exception/failure in the ToE.
When receiving input, the Notification aspect starts working as a universal debug trace logger (start logging). All collected information will be put down in a Debug Trace Log that should be analyzed and interpreted by the Monitoring Tool (of MI). *This should be implemented separately for all applications.*

Communication modules:
This aspect module (Negotiation aspect) will allow different interacting parties to define a common security policy through the use of predefined negotiation models (e.g. a simple model would be choosing the policy that has the highest security level). This will allow the different parties to collaborate in a trusted manner.

9.4 Interfaces specification

The Flinder framework and the Target of Evaluation has 4 different interfaces where the modules receive inputs.
For Flinder we distinguish two different interfaces including the Input Generator, and MFDL.
For the ToE, (see Figure 28) on I1: The ToE receives a malicious input in Binary Message Format, on I2: Prepare to receive input for injection (fuzz testing).
For more details on the input generation of Flinder framework see Chapter 11.3.4.
Output of Flinder (3): will be ToE I1, any existing interface for the basic functionality of the ToE. Output of Flinder (4): will be ToE I2, a new interface responsible for injection (fuzz testing).

9.5 Unit tests specification

Our testing module could be tested after it is weaved to other system elements in the framework, like different applications and communication modules. Unit tests will be executed on the new adapted elements of the Flinder framework including the modified Actuator and the new Network Dispatcher.

Test 1: In this test case the Actuator unit will be tested if it is capable to cooperate with the Injection aspect and provide test vectors into the interoperating applications.

Test 2: In this test case the Network Dispatcher unit will be tested if it is capable to communicate with the Notification aspect of the ToE, by sending a Flinder-modified BiMe.

10 Reaction module

10.1 Overview

This aspect module will be in charge of performing the necessary protection and mitigation strategies to keep the communication system and application safe. Security requirements of each INTER-TRUST case study will define several protection and reaction strategies that will be woven into the application. In this way it will be possible to increase the reliability and trust of the proposed security rules and the adaptability of the system to new sets of malicious behaviour and threats.

10.2 Use cases

The reaction module should be able to identify the root causes for an undesired outcome and the actions adequate to prevent recurrence. Root cause analysis helps determine what happened, how it happened, and why it happened which is one of the objective of INTER-TRUST project in the context of security failure detection. Several generic mitigation strategies will implemented within this reaction module as well as the fact it can initiate a re-negotiation mechanism. The link between cause and reaction will be specified as a reaction matrix file. According to these requirements, 2 main use cases are defined for the reaction module:

- This module should be able to determine the cause of a security policy violation
- This module should be able to call the adequate function that will activate an internal or remote action (e.g. block an IP address).

10.3 Functional specification

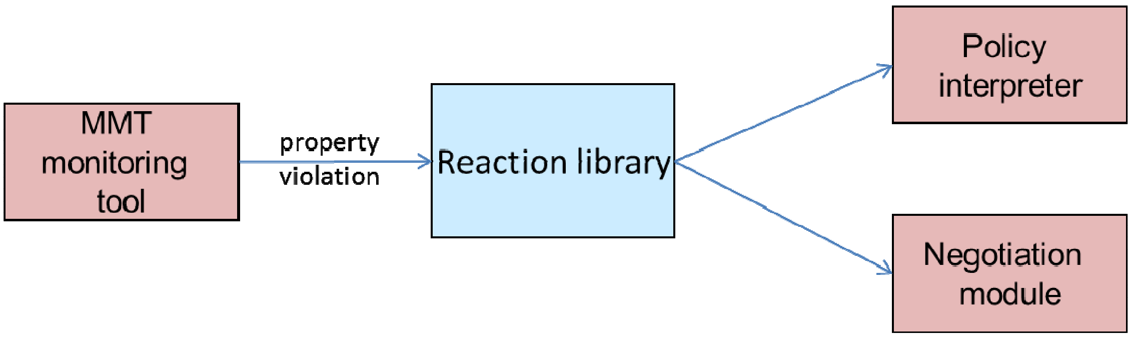


Figure 29. Conceptual view of the monitoring aspect module

Figure 29 describes the interaction between the reaction modules and the other INTER-TRUST components. The reaction modules are a set of functions stored in a library. MMT tool can activate one or several reactions if one security incident is detected. This library contains generic reactions and mitigation functions that can be called in different platforms and devices. It can easily be extended to add new application specific reactions. One of the reaction functions contacts the negotiation module to trigger a new negotiation phase.

10.4 Unit tests specification

Test 1: Deploy a specific security policy and analysis security according to a second security policy. A violation is detected and the reaction function is triggered.

11 Stand-alone monitoring and testing tools

11.1 Montimage Monitoring tool

11.1.1 Overview

MMT (Montimage Monitoring Tool) is a monitoring solution that combines: data capture; filtering and storage; events extraction and statistics collection; and, traffic analysis and reporting. It provides network, application, flow and user level visibility. Through its real-time and historical views, MMT facilitates network security and performance monitoring and operation troubleshooting. MMT's rules engine can correlate network and application events in order to detect operational, security and performance incidents. In the context of the INTER-TRUST project, Montimage will rely on MMT-Security that is a functional and security analysis tool (part of MMT) that verifies application or protocol network traffic traces against a set of MMT-Security properties. MMT-Security properties can be either "Security rules" or "Attacks" as described by the following:

- A Security rule describes the expected functional or security behaviour of the application or protocol under-test. The non-respect of the MMT-Security property indicates an abnormal behaviour.
- An Attack describes a malicious behaviour whether it is an attack model, a vulnerability or a misbehaviour. Here, the respect of the MMT-Security property indicates the detection of an abnormal behaviour that might imply the occurrence of an attack.

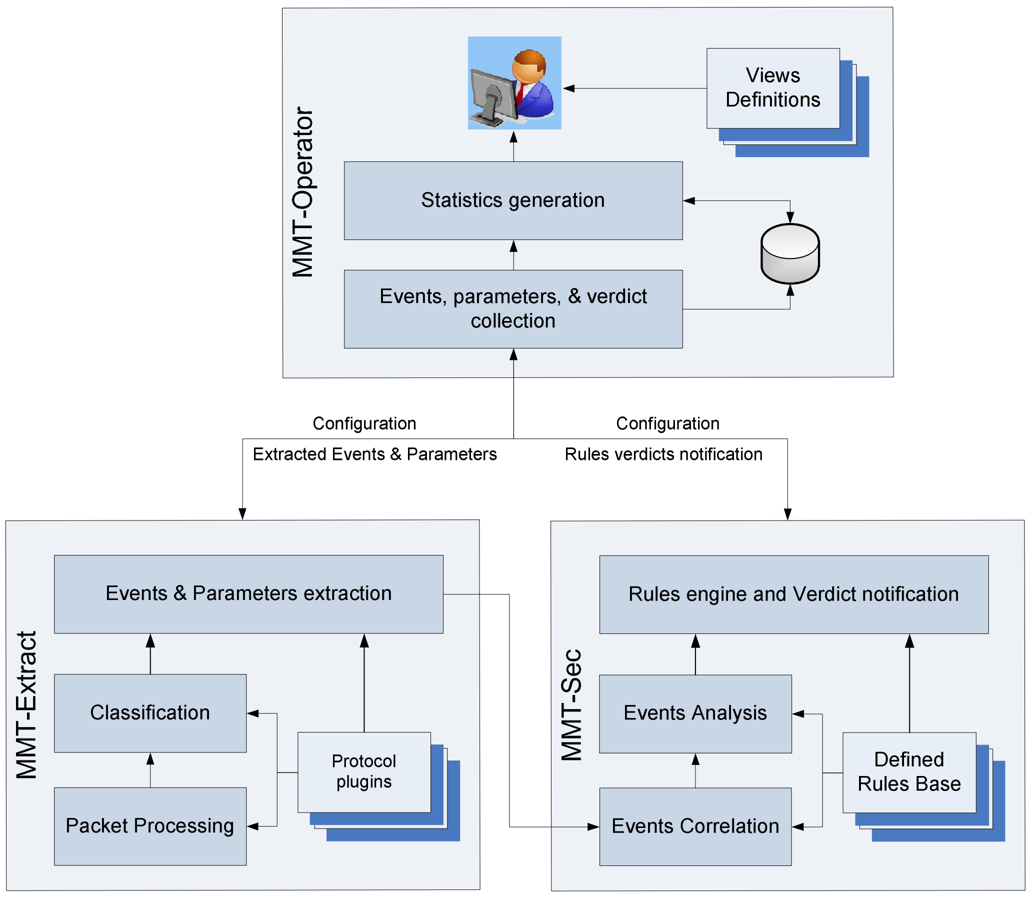


Figure 30. MMT global architecture

MMT can be delivered and installed as (1) a standalone tool that allows the analysis of live or pre-recorded network traffic or as (2) a set of two libraries for integration into third party probes. The monitoring solution can be comprised of several probes monitoring different network interfaces and application internal events with a central application correlating information to obtain a more global view of what is happening. The analysis for security purposes follows four steps:

- The definition of the monitoring architecture: This architecture depends on the nature of the system under test and its deployment in the network. Capture engines (i.e. probes) are placed at relevant elements or links in the network to obtain real time visibility of the traffic to be analysed. In the case of a distributed architecture, local traces are correlated (based on event timestamps) to obtain a broader visibility of what is going on in the network.
- The description of the system security goals and attacks based on the MMT security property format: The description specifies the security policies that the studied system has to respect (output of the negotiation mechanism) and/or the security attacks that it has to avoid. This task can be done by an expert of the system under test that understands its security requirements in details.
- The security analysis: Based on the security property specification, the passive tester performs security analysis on the captured events (application events, network packets etc.) to deduce verdicts for each property.
- The reaction: In case of a fail verdict, some reactions have to be undertaken, based on previously defined security strategies, e.g. to block any malicious behaviour or the triggering of a renegotiation mechanism.

11.1.2 Use cases

In the context of INTER-TRUST project, MMT tool will be adapted to fit the secure interoperation challenges and scenarios described in D2.1.1.

MMT should be able to capture the studied application communication (network traffic) as well as any internal application event (Business Activity Monitoring) and analyse it according to a set of security properties. These security properties can change during the analysis process because of the dynamicity of the security requirements and their change during time. The deployment of a new security should be notified to MMT so that it can respectively adapt its analysis. When a violation is detected, MMT should be able to contact the reaction module to launch a mitigation action and attenuate the security flaw. Furthermore, MMT probes are deployed in several network elements and devices and should be able to communicate to share events and data for the detection of distributed attacks.

According to these requirements, five use cases are defined for MMT tool:

- MMT should be able to capture network traffic, as well as application internal events and messages. MMT should also be able to focus on specific relevant protocols and applications by filtering irrelevant traffic.
- MMT should be able to analyse network traffic and application events according to a set of security properties (including changing security policies and security attacks and vulnerabilities).
- The initial security policies and the new ones (derived after the negotiation phase) are notified to MMT tool by the means of a specific API.
- MMT should be able to trigger some reaction after the analysis phase.

11.1.3 Functional specification

MMT is composed of 5 complementary, but independent, modules as shown in Figure 31

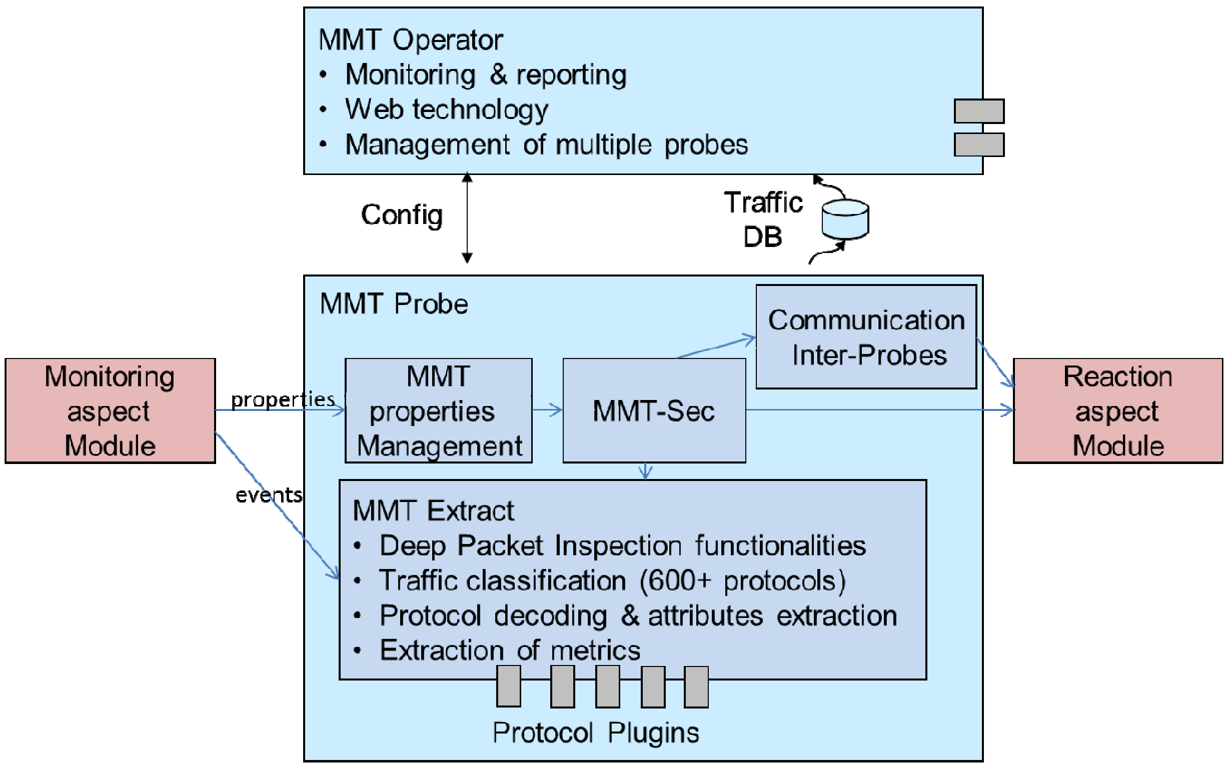

MMT Operator
• Monitoring & reporting
• Web technology
• Management of multiple probes
Config
Traffic DB
MMT Probe
Communication Inter-Probes
Monitoring aspect Module
properties
events
MMT properties Management
MMT-Sec
Reaction aspect Module
MMT Extract
• Deep Packet Inspection functionalities
• Traffic classification (600+ protocols)
• Protocol decoding & attributes extraction
• Extraction of metrics
Protocol Plugins

Figure 31. Conceptual view of the MMT tool

- MMT Properties management module receives notifications from the monitoring aspect module when a security policy is deployed in the system. This allows to MMT to update its list of security policies (i.e., adding or removing security properties) before the extraction and analysis phases.
- MMT-Extract is the core packet processing module. It is a C library that analyses network traffic using Deep Packet/Flow Inspection (DPI/DFI) techniques in order to extract hundreds of network and application based events, including: protocols field values, network and application QoS parameters and KPIs. MMT-Extract incorporates a plugin architecture for the addition of new protocols and a public API for integrating third party probes. This component already exists in the currect version of MMT. New plugins will be added to be able to manage the different INTER-TRUST case study scenarios.
- MMT-Sec is a security analysis engine based on MMT-Security properties. MMT-Sec analyzes and correlates network and application events to detect operational and security incidents. For each occurrence of a security property, MMT-Sec allows to detect whether it was respected or violated.
- The inter-probes communication module allows to ensure the communication between remotes probes. This communication is essential to share relevant events or trigger remote mitigation actions if a security property is violated.
- MMT-Operator is a visualisation application for MMT. It will allow collecting and aggregating security incidents, and present them via a graphical user interface. MMT-Operator will be customizable: the user will be able to define new views or customize from a large list of predefined ones.

11.1.4 Interfaces specification

The first input to the MMT tool is any structured data relevant for the security analysis. It can be for instance:

- Live network traffic captured by MMT probes using pcap library.
- The application internal messages notified by the monitoring aspect module.
- Any internal event (e.g., CPU usage etc.) notified by the monitoring aspect module.

The notification is based on a client connector (i.e., a generic Java library) that sends, via the network, relevant events to a local or remote MMT server. It should produce HTTP based messages in the standard format of "field: value"[9] (i.e., POST messages for sending events and GET messages pour status commands). MMT server will intercept the HTTP POST requests from the connectors and analyze the notified events relying on a dedicated plugin.

The second input of MMT is also notified by the monitoring aspect module after each dynamic deployment of security policies. These security policies are notified to MMT (in an XML format to be defined as a second step).

After the analysis of the security policies, MMT probe can trigger local reactions or contact remote probes to activate remote reactions. More details about this interface are provided in section 10.

11.1.5 Unit tests specification

Test 1: The application deployes a security policy and notifies it to MMT. MMT analyzes network traffic and determines that no security property is violated.

Test 2: The application notifies relevant internal events to MMT. MMT analyses them according to specific security policies (dealing with internal messages). No security property violation is detected.

Test 3: The application deployes a security policy and does not notify the change to MMT. MMT detects the violation of a set of security policies and launches some predefined mitigation action.

11.2 Active testing tool

11.2.1 Active Testing Approach

Active Testing consists of applying a set of test scenarios on a System Under Test (SUT) to check its conformance according to its specified requirements. It is one of the most important phases of the system building, the most expensive and time-consuming. Systems are error-prone and the main goal of testing is identifying the errors in a system implementation.

Ideally, the testing phase cannot be concluded until the whole system is analyzed, all the errors identified and corrected. In practical terms, the real question is: is it possible to test a whole system?

- A first drawback is how to generate all possible test cases. For a human being, and dealing with a complex system, it is impossible to think through all possible situations, single and combined, to generate a complete test sequence, even worst, executing them.

[9] RFC2822.

- A second limitation is the time: it is not straightforward to determine how long the process of testing a system will take. The experience and studies have shown that testing may consume up to 50% of the time and the project resources [5] in a system development. When errors are found, it is expected that the implementation is changed, and the tests redone. Not only the tests that had previously detected faults, but, ideally, all set of tests should be re-executed in order to be sure that new errors were not introduced. This scenario may lead a system development to take longer time that previously expected.

In industrial applications, *time-to-market* is one of main reasons for reducing the time of the system development. Meanwhile, even when timing requirements or other safety aspects are not taken into account the selection of which tests should be applied is not a direct process. These contexts bring other practical questions, such as:

- Is it possible to reduce the time spent in testing?
- How to select the test cases that must be generated and executed?

Automation of testing activities seems to get around this problem. Automation may help in making the testing process faster, in making it less susceptible to errors and more reproducible [5].

In a common active testing approach there are two main steps:

- The (automatic) generation of a set of test cases based on the system specification.
- The processing of these test cases on the implementation. The conformance verdict is deduced after the analysis of the system reaction to these stimuli (tests).

The strength of the active testing is in ability to focus on particular aspects of the implementation. Indeed, the test cases can be for instance limited to a specific type of errors, or to an important state of the system. On the other hand, the test case choice and production can turn out to be complicated. Besides, testing actively a system can disturb its normal functioning. For this reason, active testing is usually performed on a system implementation before its commercialization.

Figure 32 describes the steps for the active testing general methodology. Based on a formal specification of the system under test, we try to generate automatically a set of test cases that we provide to the active tester unit. This tester will apply the test scenarios on the system implementation to stimulate it and collect then its reaction (output messages). These messages are analysed: the tester checks whether they correspond to the desired outputs described in the specification. If it is the case, the verdict would be `pass', otherwise, it is `fail'. In some cases (for instance, when the system is non-deterministic), the verdict can be `inconclusive'.

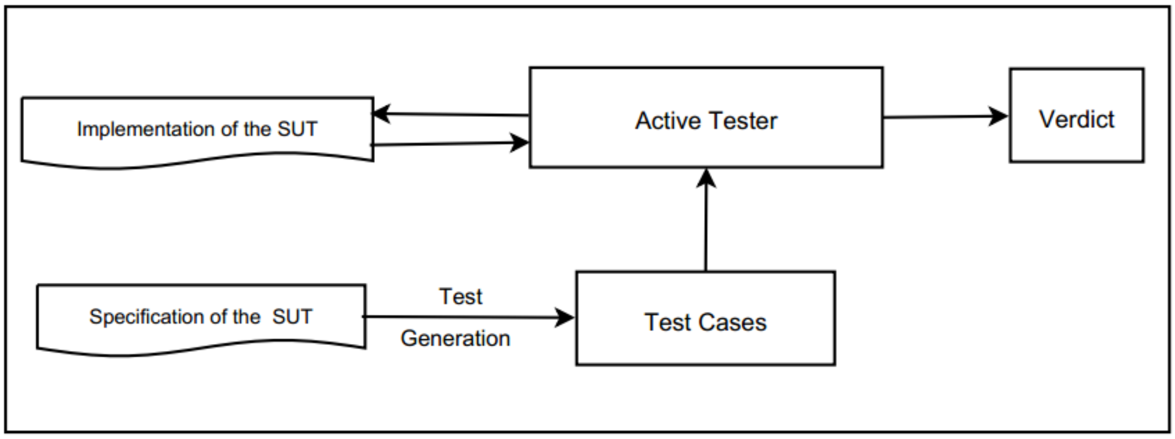


Figure 32. Active testing methodology

Automated Test Generation: In practice, test generation is in general performed by humans after studying the specification [6]. In formal contexts, a model represents the system specification and it allows an automatic test case generation. Model-based testing is a technique where a model is considered as an input to generate test suites based on dedicated algorithms. These algorithms allow the process of generating and selecting test cases.

In a test generation process, there are two important concepts that must be considered, that are *soundness* and *completeness*. We define these concepts based on [7]:

- Soundness: Generated test cases should never cause a fail verdict when executed on a correct implementation.
- Completeness: For each possible implementation that does not conform to the specification model, it is possible to generate a test case that causes a fail verdict with respect to that system under test (SUT).

A number of methods have been proposed for test case generation. In the following we will present the TestGen-IF tool.

11.2.2 TestGen-IF tool

The TestGen-IF [8] tool is based on the IF-2.0 simulator [9] that allows to construct an accessibility graph from an IF specification. It implements an automated test generation algorithm based on a Hit-or-Jump exploration strategy [10] and a set of test purposes. In Figure 33, we show the basic architecture of the TestGen-IF tool.

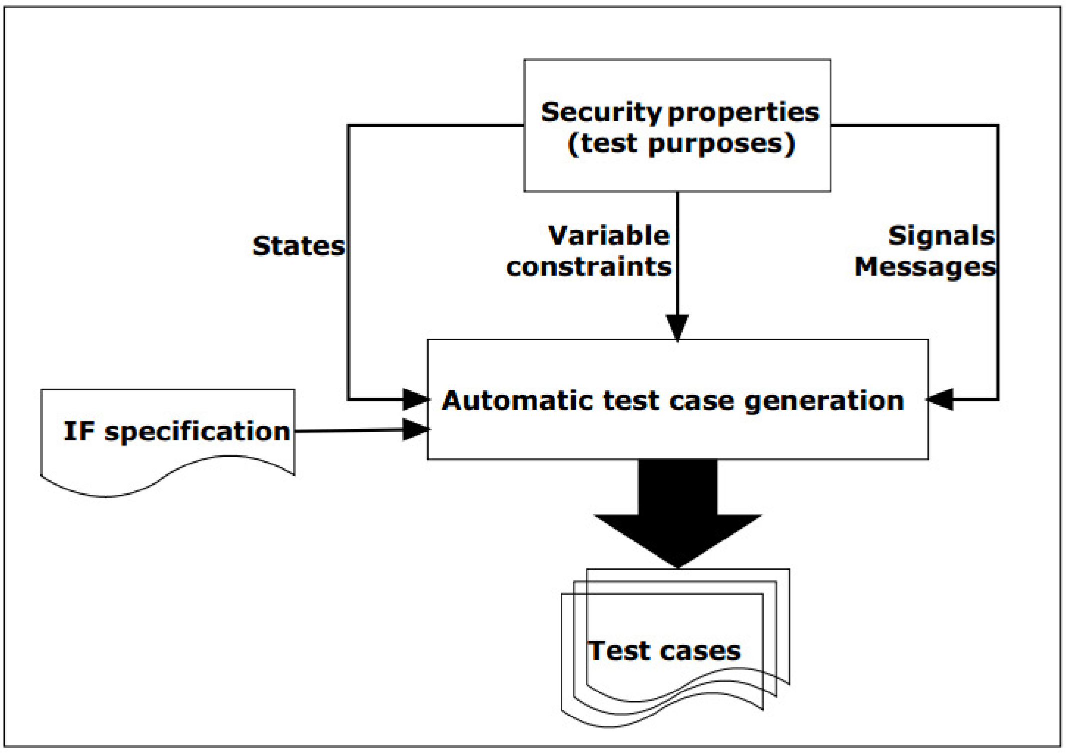


Figure 33. The basic architecture of TestGen-IF

From the current state, the tool constructs a partial accessibility graph. The partial graph is constructed using breadth first search or depth first search algorithm. The depth of the partial graph can be specified in the tool inputs. At any moment it conducts a local search from the current state in a neighbourhood of the partial graph. If a state is reached and one or more test objectives are satisfied (a Hit), the set of test objectives is updated and a new partial search is conducted from this state. Otherwise, a partial search is performed from a random graph leaf (a Jump). The advantages of TestGen-IF tool is that it efficiently constructs tests sequences with high fault coverage, avoiding the state explosion and deadlock problems encountered respectively in exhaustive or exclusively random searches.

A test objective is a set of ordered conditions. This means that the conditions have to be satisfied with the order that is given by the test case. A condition is a conjunction of a process instance constraint, state constraint, action constraints and variable constraints. A process instance constraint indicates the identifier of a process instance. An action constraint describes sending or receiving messages. A variable constraint gives conditions on variable values. A test objective specifies the property to be validated in the system implementation. The test case generation algorithm fetch for this property in the reachability graph that describes the joint behaviour of the two systems. When the property is satisfied, the algorithm generates the path that leads to this property. This path is transformed to an executable test case to be applied later to the system implementation.

11.3 Fuzz testing tool

11.3.1 Overview

Flinder was originally built around the same concept as the INTER-TRUST project is aiming at: in a plug-in-able architecture it looks for certain well-defined types of flaws which may lead to security vulnerabilities. This is supported by Flinder both in black-box- and white-box-mode.
In this chapter in the first section first fuzz testing will be described in general, in the second part the basic operation of Flinder will be explained then illustrated with an example about JPEG file format fuzzing.

A preliminary requirement list is presented for the future integration of Flinder tool to INTER-TRUST framework.

11.3.1.1 Fuzz testing description in general

Vulnerability detection through fuzz testing [11]

Common security vulnerabilities may be detected in various ways, from source code analysis to black-box testing. As source code analysis is a design-time technique and we are evaluating the external interfaces of composite services, we focus on detecting vulnerabilities through the use of active testing methods – specifically fuzz testing.

Most traditional security testing methodologies focus on finding evidence of known vulnerability types. Fuzz testing approaches the same problem from a different angle – instead of trying to locate specific vulnerabilities, it attempts to manipulate the input to the target composite service (the Target of Evaluation – ToE) in order to discover previously unknown typical vulnerabilities.

Fuzzing is a highly automated testing technique that covers numerous boundary cases using invalid data (from files, network protocols, API calls and other targets) as application input to better ensure the absence of exploitable vulnerabilities. The name comes from modem applications' tendency to fail due to random input caused by line noise on "fuzzy" telephone lines. Fuzzing is not purely a security testing technique – it has numerous applications in quality assurance (QA) processes as well (specifically robustness testing). Even there it focuses on identifying bugs arising from input validation issues, and does not validate the actual business logic implementation; fuzz testing cannot substitute for functional testing. Fuzz testing also tends to be binary in the sense that it only evaluates whether a certain test vector causes the ToE to crash, lock up, or act in a way that is easily identifiable as non-conformant – it typically does not attempt to identify the cause of the crash in detail. As this also means that typical fuzz testing produces no false positives.

We can categorise fuzz testing approaches in several ways:

- Smart vs. dumb testing: whether the fuzzer is aware of the input's inner structure
- Serial vs. parallel testing: whether a smart fuzzer modifies one field at a time, or multiple fields
- Handling of data relationships: whether a smart fuzzer can handle data relationships such as length/count fields, checksums, encryption, and compression

- Randomness of test vectors: whether the fuzzer creates its test cases randomly, or in a structured deterministic way
- Generation of test vectors: whether the fuzzer generates test cases on its own, or whether it modifies existing (valid) input
- Test data creation: whether a fuzzer generates test vectors randomly, iterates through hard-coded values, generates test cases in a configurable way from a pre-set pool, or adapts the test vectors during the test itself
- **Black-box / white-box / grey-box: whether the tester has access to the ToE's source code**

Black-box, white-box and grey-box fuzzing [12]

Fuzzing can be performed with or without additional knowledge about the ToE (such as specifications of internal interfaces and source code). Depending on the availability of such data and where the fuzzing itself takes place within the ToE, we can define black-box, white-box and grey-box fuzz testing scenarios.

Fuzzing is traditionally a black-box process. In a black-box test the only necessary information is a description of the ToE's external interfaces, and test cases are generated to cover the possible input space as well as possible. Black-box fuzzing does not need special access to the ToE, and is much easier to automate than the other two types.

White-box fuzzing is done against the internal interfaces of the ToE *at design time* by injecting errors directly into the target program's function inputs via DLL hooking and other similar techniques. It requires tight integration with the ToE (including modification of the source code so that the fuzzer can perform fault injection), and the fuzzing test vectors need to be generated against the internal functions of the ToE. If a fuzzer discovers a defect via white-box testing, it can typically give more useful information than a black-box fuzzer to the developer about how to fix it.

Grey-box fuzzing is similar to black-box testing in that it tests the external interfaces of the ToE; however, it is assisted by additional knowledge of the ToE's source code, which the fuzzing engines can use to optimise the generated test cases and maximise code coverage. For example, the SAGE (Scalable, Automated, Guided Execution) fuzzer defines input constraints based on branches in the source code to reduce the number of necessary test cases and increase efficiency, along with much greater code coverage. Similarly, the Evolutionary Fuzzing System (EFS[10]) is a grey-box system that attempts to increase code coverage by specifying targets in the source code and using genetic algorithms to eventually find inputs that will end up inside the target area.

Fuzz testing input-methods[13][14]

[10] http://www.vdalabs.com/tools/efs_gpf.html

A crucial aspect of fuzz testing is specifying the inputs that are worthwhile to modify along with defining rules for the modifications themselves. We can define three different fuzz testing methods that each deal with this challenge in different ways:

Random fuzzing is the simplest technique. A random fuzzer generates test vectors from random input restricted by several simple rules (such as 'alphanumeric characters only'). Random fuzzers are extremely easy to implement for any sort of system and require no integration, but their results are hard to reproduce and produce very few useful test cases; most output from a random fuzzer can be expected to be discarded by the ToE as invalid.

Mutation-based fuzzing can be considered a more refined version of random fuzzing. Instead of generating purely random input, a mutation-based fuzzer can modify previously-captured input repeatedly to generate test vectors. These modifications can consist of appending random bit streams to existing input, or randomly modifying bits in such input so as to simulate a noisy channel; all modifications can also be optionally restricted by simple rules. Mutation-based fuzzers are generic and can be used for a wide array of ToEs without the need for extensive adaptation; however, as they are modifying valid input, they need to be integrated with the ToE. Mutation-based fuzzers are much more likely to discover vulnerabilities than random fuzzers, but their output is still unpredictable and likely to fail if there are dependencies between fields of the protocol or file format.

Model-based (deterministic) fuzzing is a substantially different approach. A model-based fuzzer requires a model describing the format of the ToE's input, along with the modifications used to create the test vectors from the original input. A model-based fuzzer can modify previously-captured input (or a series of inputs), or it can attempt to construct completely new input on its own, essentially generating all possible sentences of the grammar defined by the fuzzing models; both of these approaches are deterministic. Such a fuzzer is necessarily *specific* to the ToE and the type of input necessary for the ToE; however, the modification algorithms are generic enough to be reusable. This method's efficiency depends on the completeness of the model and the input used as the basis of the fuzzing. Model-based fuzzing is the most likely of all fuzzing methods to discover vulnerabilities, but it requires substantial effort to be invested in order to evaluate a certain ToE. **Flinder is a model-based fuzzer.**

In model-based fuzzing (and the other two methods as well, to a certain degree) the actual fuzzing process depends on pre-written rules; the biggest challenge when performing fuzz testing is identifying rules for a certain ToE that minimize the number of generated test vectors and maximize the likelihood of detecting vulnerabilities. For example, if the ToE is a media player, it is *reasonably safe* to assume that most metadata fields in media files are going to be processed in the same way, and therefore fuzzing all of them is not necessary. Similarly, when modifying a 32-bit integer field, it is not necessary to test for all possible values of a 32-bit integer; the modification algorithm should instead focus on boundary values and use a step value to skip over most other 'uninteresting' numbers. The definition of such fuzzing rules is necessarily a manual process. If the source code and development documentation of the ToE is available (*white-box* testing), it makes the definition of fuzzing rules easier.

A fuzz tester can discover whether a certain input triggered a vulnerability by checking the output from the ToE or any unexpected actions done by the ToE (e.g., crash, lock-up). This requires the implementation of a *test harness* that is specific to the ToE and the fuzzer.

11.3.1.2 About Flinder

Flinder is an automated **security and robustness testing tool** developed by SEARCH-LAB for detecting typical security-relevant programming bugs. By automatically executing a vast number of security and robustness tests, Flinder can greatly increase the overall security properties of a system, since it can detect most occurrences of certain types of typical security-relevant programming bugs causing a large amount of exploitable vulnerabilities.

- **Fuzzing**: Security-relevant test vectors are generated by manipulating correct input messages in a systematic way in order to test values that potentially induce security-relevant programming bugs.
- **Support for different message formats**: Besides some natively supported formats (e.g. XML, ASN.1 DER), Flinder can parse arbitrary binary messages with the help of message format descriptions (MFDLs) specifying the structure and the encoding. Testers only need to provide these MFDL descriptions to Flinder to be able to execute security tests.
- **Reactively iterating test algorithms:** Flinder can observe the ToE's reaction to the test vectors and generate the next test cases based on the responses received for the previous messages (e.g. a successively approximating test algorithm can be implemented this way).
- **Cryptographic and encoding support**: Flinder can also handle (decode, modify and then re-encode) encrypted, digitally signed or compressed messages.
- **Protocol state machine**: State transitions of complex protocols are tracked by an internal state machine defined by a UML state-chart.
- **Automatic test report generation**: The results of the executed tests are collected in an easy-to-navigate hypertext output file

11.3.2 Use-case

The use-case for Flinder fuzz-testing tool will be to be able to find as many implementation errors as possible by automatically generated test cases.

11.3.3 Functional Specification

For the convenience of the reader, find an illustration In Figure 34 below, that shows the main modules of the current version of Flinder, and the connections between them. This architecture shall be adapted to the INTER-TRUST Framework. (For other detailed information, see “Basic operation” of Flinder at section 9.1.1).

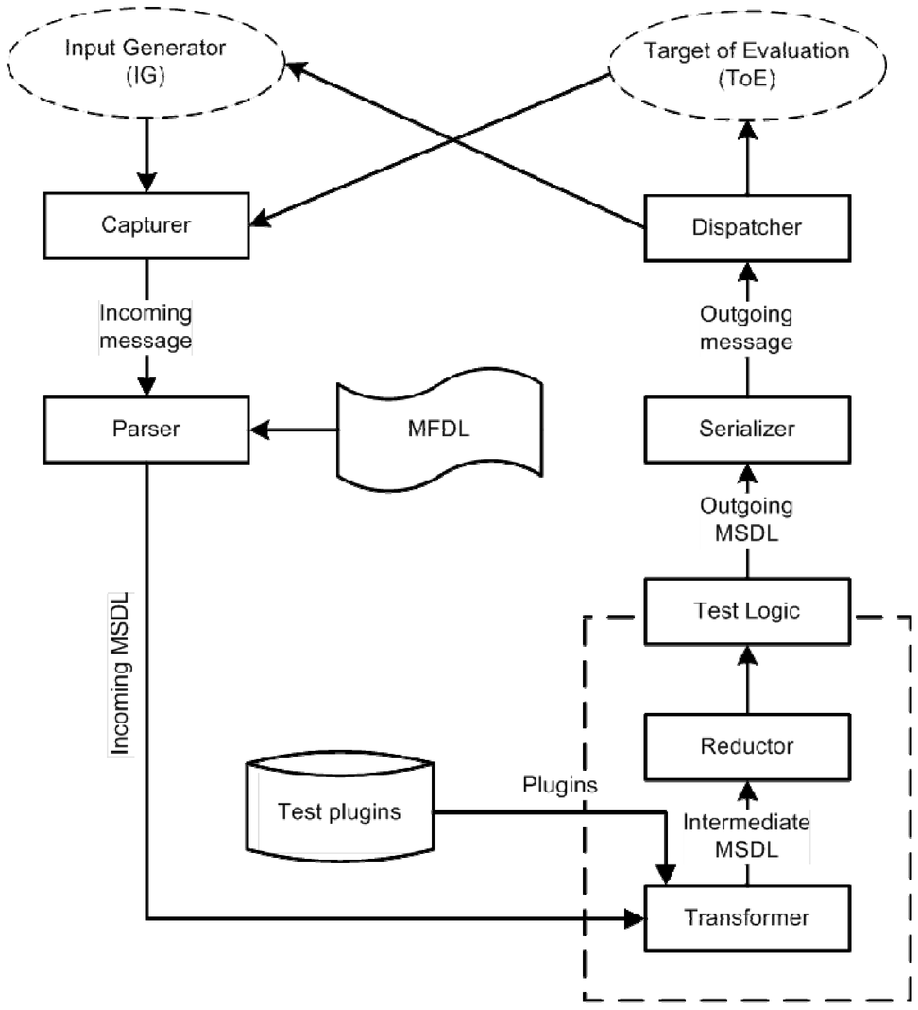
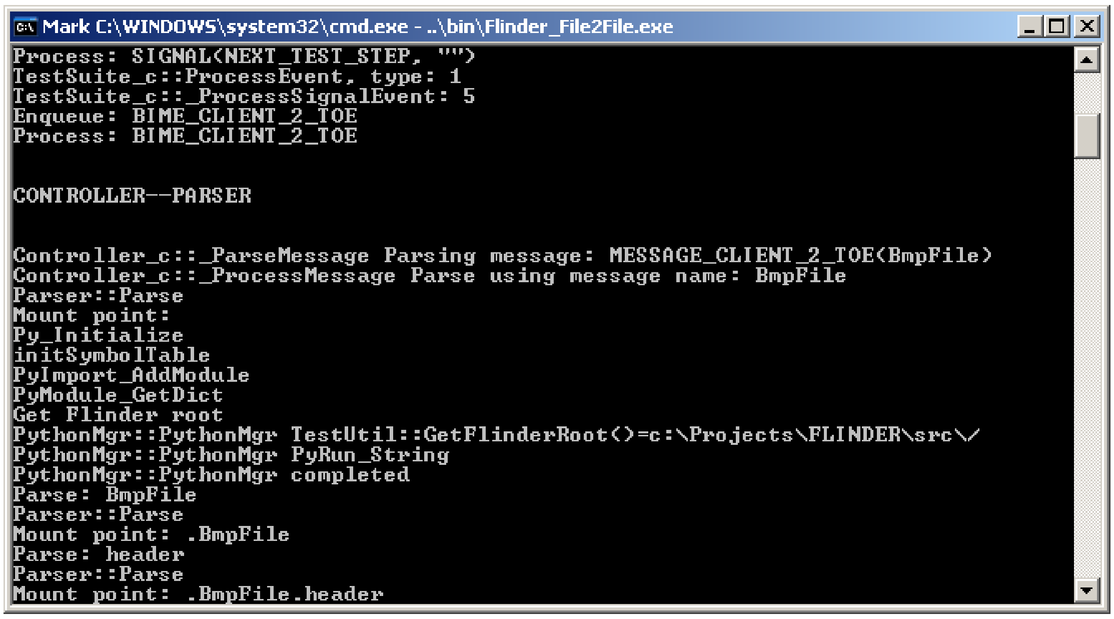


Figure 34 Flinder Basic operation

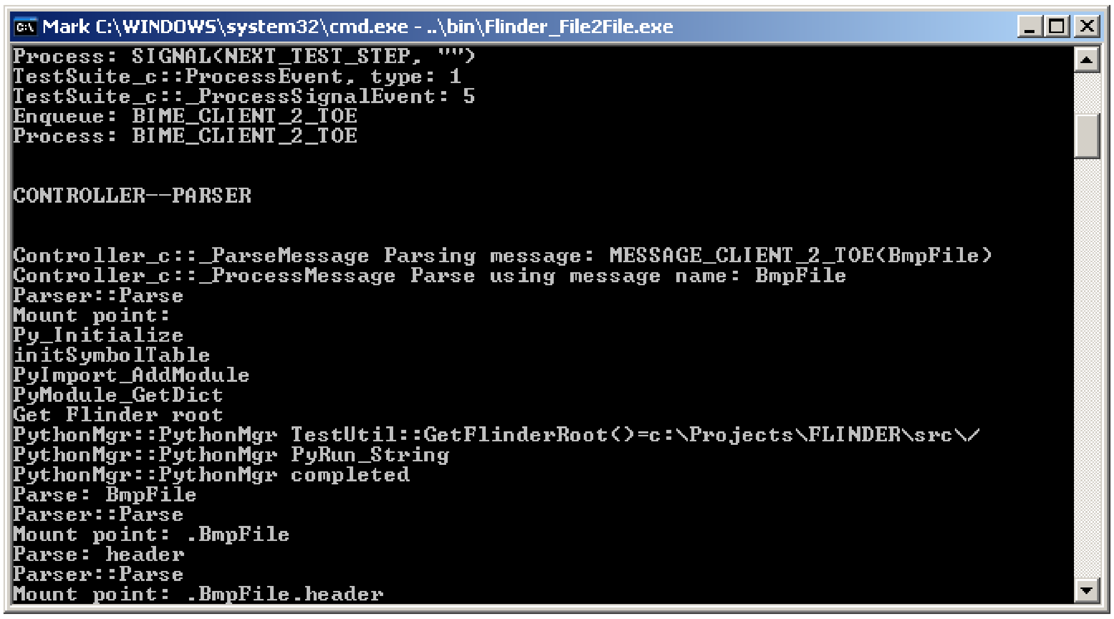
The two screenshots below (Figure 35) demonstrate Flinder’s operation; the first screenshot was taken during the parsing process, the second one after the serialisation of a modified message. In both cases, the message was a BMP file.


Mark C:\WINDOWS\system32\cmd.exe - ..\bin\Flinder_File2File.exe

```
Process: SIGNAL(NEXT_TEST_STEP, "")
TestSuite_c::ProcessEvent, type: 1
TestSuite_c::_ProcessSignalEvent: 5
Enqueue: BIME_CLIENT_2_TOE
Process: BIME_CLIENT_2_TOE


CONTROLLER--PARSER


Controller_c::_ParseMessage Parsing message: MESSAGE_CLIENT_2_TOE(BmpFile)
Controller_c::_ProcessMessage Parse using message name: BmpFile
Parser::Parse
Mount point:
Py_Initialize
initSymbolTable
PyImport_AddModule
PyModule_GetDict
Get Flinder root
PythonMgr::PythonMgr TestUtil::GetFlinderRoot()=c:\Projects\FLINDER\src\/
PythonMgr::PythonMgr PyRun_String
PythonMgr::PythonMgr completed
Parse: BmpFile
Parser::Parse
Mount point: .BmpFile
Parse: header
Parser::Parse
Mount point: .BmpFile.header
```

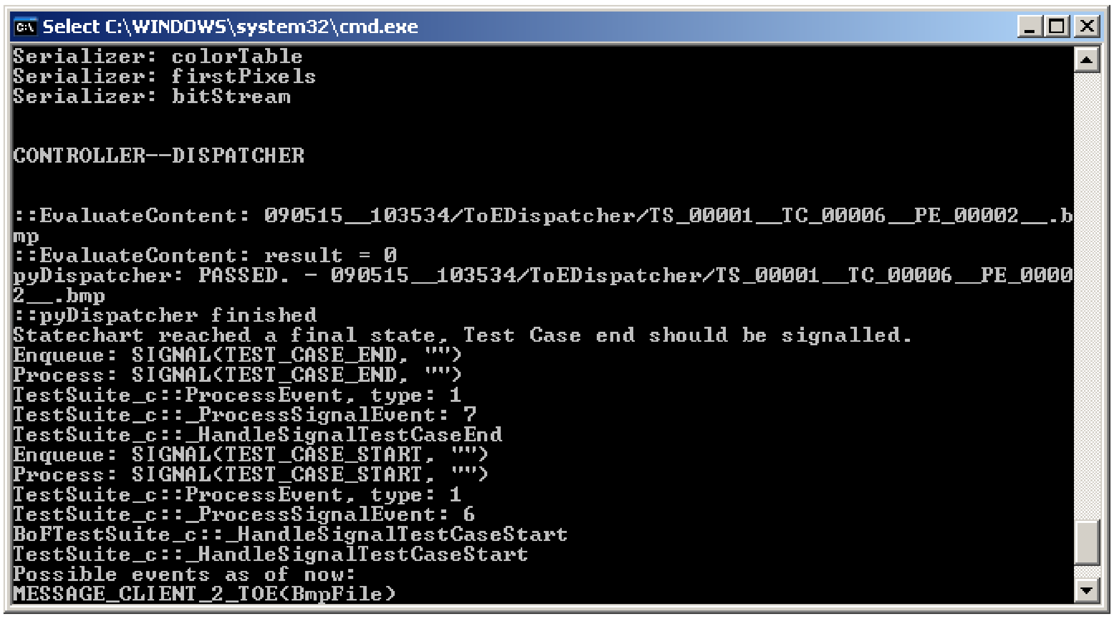

Mark C:\WINDOWS\system32\cmd.exe - ..\bin\Flinder_File2File.exe

```
Process: SIGNAL(NEXT_TEST_STEP, "")
TestSuite_c::ProcessEvent, type: 1
TestSuite_c::_ProcessSignalEvent: 5
Enqueue: BIME_CLIENT_2_TOE
Process: BIME_CLIENT_2_TOE


CONTROLLER--PARSER


Controller_c::_ParseMessage Parsing message: MESSAGE_CLIENT_2_TOE(BmpFile)
Controller_c::_ProcessMessage Parse using message name: BmpFile
Parser::Parse
Mount point:
Py_Initialize
initSymbolTable
PyImport_AddModule
PyModule_GetDict
Get Flinder root
PythonMgr::PythonMgr TestUtil::GetFlinderRoot()=c:\Projects\FLINDER\src\/
PythonMgr::PythonMgr PyRun_String
PythonMgr::PythonMgr completed
Parse: BmpFile
Parser::Parse
Mount point: .BmpFile
Parse: header
Parser::Parse
Mount point: .BmpFile.header
```

Figure 35. Parsing a BMP file



```
Serializer: colorTable
Serializer: firstPixels
Serializer: bitStream


CONTROLLER--DISPATCHER


::EvaluateContent: 090515__103534/ToEDispatcher/TS_00001__TC_00006__PE_00002__.b
mp
::EvaluateContent: result = 0
pyDispatcher: PASSED. - 090515__103534/ToEDispatcher/TS_00001__TC_00006__PE_0000
2__.bmp
::pyDispatcher finished
Statechart reached a final state, Test Case end should be signalled.
Enqueue: SIGNAL(TEST_CASE_END, "")
Process: SIGNAL(TEST_CASE_END, "")
TestSuite_c::ProcessEvent, type: 1
TestSuite_c::_ProcessSignalEvent: 7
TestSuite_c::_HandleSignalTestCaseEnd
Enqueue: SIGNAL(TEST_CASE_START, "")
Process: SIGNAL(TEST_CASE_START, "")
TestSuite_c::ProcessEvent, type: 1
TestSuite_c::_ProcessSignalEvent: 6
BoFTestSuite_c::_HandleSignalTestCaseStart
TestSuite_c::_HandleSignalTestCaseStart
Possible events as of now:
MESSAGE_CLIENT_2_TOE(BmpFile)
```

Figure 36 Serialising and dispatching a modified BMP file

Flinder creates multiple HTML report files after testing. It generates a report file for each **test group**[11], and it creates a higher-level summary about all test groups combined afterwards. In case of adaptation to INTER-TRUST Framework all Debug Trace Log will be handed over to the Monitoring Tool (by MI), that could monitor and analyze the given test results.

[11] In Flinder, a ‘test group’ refers to a subset of fields that are being modified during the testing process (either sequentially or in parallel).

11.3.4 Interfaces specification

The Flinder framework and the Target of Evaluation have 4 different interfaces where the modules receive inputs.

- Input generation for Flinder (1): receives like policy negotiation messages or other messages from normal operation of the interoperating applications relevant for the use case.
- Input generation for Flinder (2): Flinder's XML-based MFDL format – as seen on Figure 37

```
<mfdl>
     <QuickTimeFile encoding="Binary">
     <OuterAtoms type="all" msdl:childBufferSize="1">
          <preParseAction>
               top.childBufferSize = "%s" % len(parseBuffer)
          </preParseAction>
          <OuterAtom>
               <choice>
                    <preParseAction>
                         s = parseBuffer[4:8]
                         choice.value = 'Unknown'
                         if s == 'free': choice.value = 'Free'
                         if s == 'wide': choice.value = 'Wide'

                         if s == 'mdat': choice.value = 'MovieData'
                         if s == 'moov': choice.value = 'Movie'
                    </preParseAction>
                    <Unknown type="UnknownAtom"/>
                    <Free type="FreeAtom"/>
                    <Movie type="MovieAtom"/>
                    <Wide type="WideAtom"/>
                    <MovieData type="MovieDataAtom"/>
               </choice>
          </OuterAtom>
     </OuterAtoms>
...
</mfdl>
```

Figure 37. Flinder MFDL fragment

The definition of fields to be fuzzed is a more intuitive process, and depends on the particular file format or protocol as well as additional information about how the ToE processes such input.

For the Output of Flinder see Chapter 9.4.

11.3.5 Unit tests specification

According to the new design- adaptation of Flinder to INTER-TRUST Framework - See more details at section 9.3: Functional specification. For the convenience of the reader please see the modified elements of Flinder architecture in the Figure below.

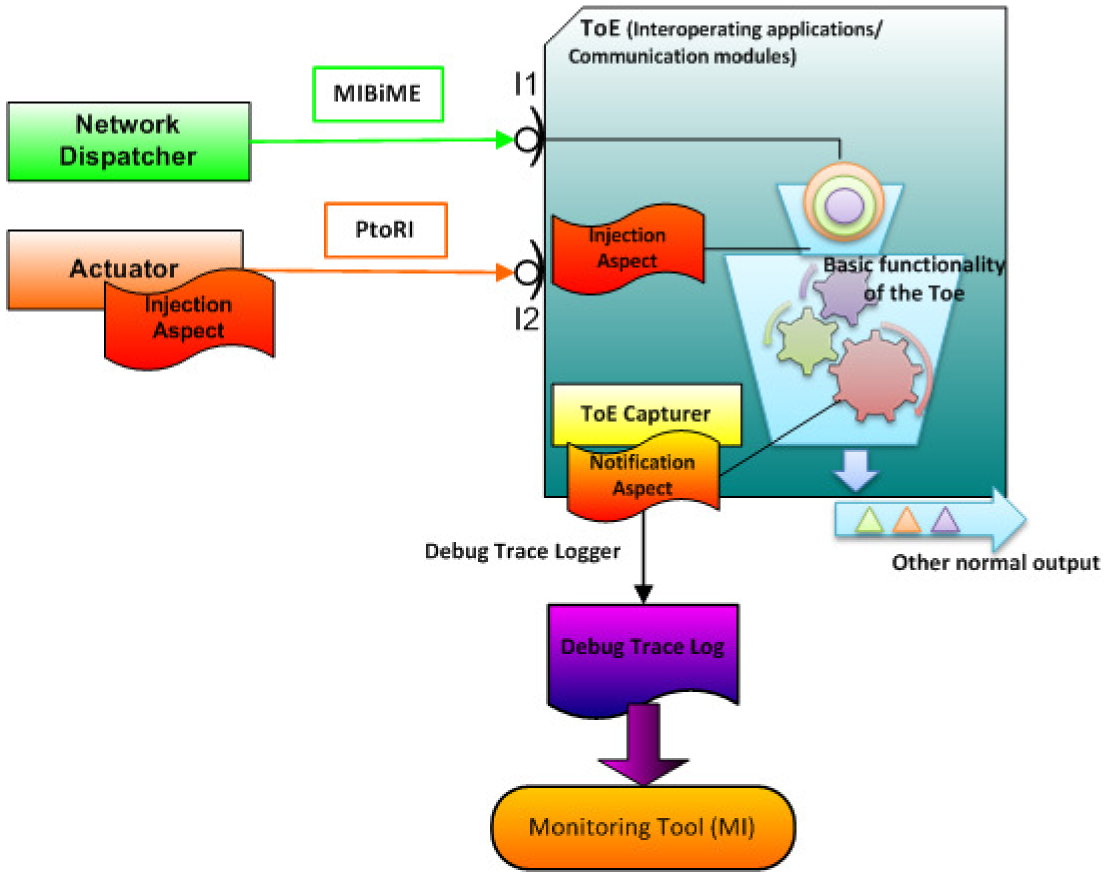


Figure 38. Modified elements of Flinder architecture

Unit tests will be executed on the:

- Modified Actuator module: In this test case the Actuator unit will be tested if it is capable to cooperate with the Injection aspect and provide test vectors into the interoperating applications.
- Network dispatcher module: In this test case the Network Dispatcher unit will be tested if it is capable to communicate with the Notification aspect of the ToE, by sending a Flinder-modified BiMe.

12 Conclusion

This document presents a first version of the specification and design of the secure interoperability framework and tools that constitutes the INTER-TRUST solution. This framework supports security mechanisms that allow secure interoperation between different parties, and dynamically adapting the security policies that drive that interaction due to changes on the running environment. The validation of the security requirements is also supported in the framework using dedicated testing and monitoring tools. The implementation task will rely on this specification and the AOP framework selected in task T4.1 to deliver a first software solution to integrate it and evaluate it in the context of INTER-TRUST case studies (Vehicle to Infrastructure communication, Vehicle to Vehicle communication and electronic voting). The second version of this document will present more detailed specification of the INTER-TRUST framework solution.

References

[1] Moses, T. (Ed.) OASIS eXtensible Access Control Markup Language (XACML) Version 2.0, OASIS Standard, 2005.

[2] SEMIRAMIS. Secure Management of Information Across Multiple Stakeholders. http://www.semiramis-cip.eu.

[3] DESEREC. Dependability and Security by Enhanced Reconfigurability. http://www.deserec.eu.

[4] C. Basile, A. Lioy, G.M. Perez, F.J.G. Clemente, A.F.G. Skarmeta. POSITIF: A Policy-Based Security Management System. Policies for Distributed Systems and Networks, 2007. POLICY '07. Eighth IEEE International Workshop on , vol., no., pp.280, 13-15. 2007.

[5] G.J. Tretmans and A. Belinfante. Automatic Testing with Formal Methods . In 7th European Int. Conference on Software Testing, Analysis & Review, EuroSTAR'99. Barcelona, Spain.

[6] J. Tretmans. Testing Techniques. Lecture Notes. Formal Methods and Tools Group, Faculty of Computer Science, University of Twente, 2001.

[7] A. Belinfante and L. Frantzen and C. Schallhart. Tools for Test Case Generation. In Model-based Testing of Reactive Systems: Advanced Lectures. Springer-Verlag, 2005.

[8] Ana Rosa Cavalli, Edgardo Montes De Oca, Wissam Mallouli, and Mounir Lallali. Two complementary tools for the formal testing of distributed sys-tems with time constraints. In Proceedings of the 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, DS-RT '08. Washington, DC, USA, 2008. IEEE Computer Society.

[9] M. Bozga, S. Graf, I. Ober, I. Ober, and I. Sifakis. The if toolset. In Lecture Notes in computer Science, volume 3185, pages 237–267. Springer, 2004.

[10] Ana R. Cavalli, David Lee, Christian Rinderknecht, and Fatiha Zaidi. Hit-or-jump: An algorithm for embedded testing with applications to in services. InInternational Conference on Formal Techniques for Networked and Distributed Systems, volume 156, pages 41–56, 1999.

[11] Software Engineering Institute, Software Assurance Curriculum Project Volume I: Master of Software Assurance Reference Curriculum, Technical Report, August 2010. http://www.cert.org/archive/pdf/10tr005.pdf

[12] Oehlert P.: Violating Assumptions with Fuzzing. IEEE Security & Privacy vol 3, issue 2, pp. 58-62. doi: 10.1109/MSP.2005.55 (2005)

[13] Takanen A., DeMott J., Miller C.: Fuzzing for Software Security Testing and Quality Assurance. Artech House, Inc. (2008) ISBN: 978-1-59693-214-2

[14] Godefroid P., Kiezun A., Y. Levin M.: Grammar-based whitebox fuzzing. Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation (PLDI '08), pp. 206-215. doi:10.1145/1375581.1375607 (2008)

[15] Badii A, User-Intimate Requirements Hierarchy Resolution Framework (UI-REF): Methodology for Capturing Ambient Assisted Living Needs, Proceedings of the Research Workshop, Int. Ambient Intelligence Systems Conference (AmI'08), Nuremberg, Germany November 2008.

[16] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel et G. Trouessin. Organization Based Access Control. IEEE 4th International Workshop on Policies for Distributed Systems and Networks (Policy 2003), Lake Come, Italy, June 4-6, 2003