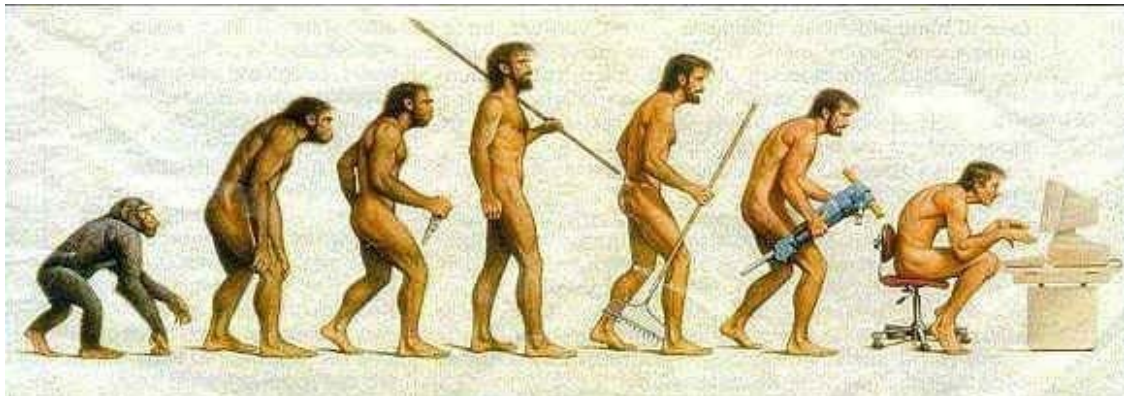




SERIOUS

DELIVERABLE

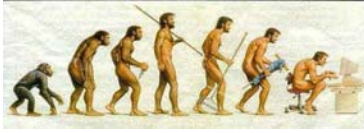
D3.6 – Definition of UML models/interface enhancements



Project number: ITEA 04032
Document version no.: WP3 Deliverable 3.6 Final Version
Edited by: Bertin, Softeam, Surlog

ITEA Roadmap domains:
Major: Services & software creation

ITEA Roadmap categories:
Major: Software engineering
Minor: System engineering



HISTORY

Document version #	Date	Remarks
V0.1	11/09/06	First draft version
V0.2	15/07/07	Final draft version
V1	03/10/07	Final version

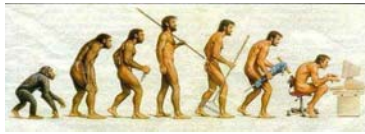


TABLE OF CONTENTS

1	INTRODUCTION	5
1.1	Safety profile.....	5
1.2	Security profile	5
2	NOTATIONS	7
3	SAFETY PROFILE	8
3.1	Use case diagram.....	8
3.1.1	Actor	8
3.1.1.1	Human.....	8
3.1.1.2	No human.....	9
3.1.2	Use case	9
3.1.2.1	Critical.....	9
3.2	Component diagram	11
3.2.1	Component.....	11
3.2.1.1	Reliability.....	11
3.2.1.2	Maintenability	11
3.2.1.3	Robustness	11
3.3	Class diagram	12
3.3.1	Class.....	12
3.3.1.1	Reliability.....	12
3.3.1.2	Maintenability	12
3.3.1.3	Robustness	13
3.4	Sequence diagram	14
3.4.1	Message.....	14
3.5	State chart diagram	15
3.5.1	State.....	15
3.5.1.1	Reliable	15
3.5.1.2	Error	15
3.5.1.3	Failure	16
3.5.2	Transition	16
3.5.2.1	Fault	16
4	SECURITY PROFILE	17
4.1	Use case diagram.....	17
4.1.1	Actor	17
4.1.1.1	Trusted	17



4.1.1.2	Untrusted	18
4.1.1.3	External	18
4.1.2	Use case	19
4.1.2.1	Access control.....	19
4.1.2.2	Security	20
4.1.3	Association.....	20
4.1.3.1	Guarded access.....	20
4.2	Component diagram	22
4.2.1	Component	22
4.2.1.1	Access control.....	22
4.2.1.2	Security	23
4.2.2	Interface	23
4.2.2.1	Access control.....	23
4.2.2.2	Security	24
4.2.3	Association.....	24
4.2.3.1	Provided	24
4.2.3.2	Required.....	24
4.2.3.3	Guarded access.....	25
4.3	Class diagram	26
4.3.1	Class.....	26
4.3.1.1	Secure.....	26
4.3.1.2	Interface	27
4.3.1.3	Access control.....	27
4.3.1.4	Security control.....	27
4.3.2	Association.....	28
4.3.2.1	Guarded access.....	28
4.3.3	Association class.....	28
4.3.3.1	Security arbitrator	28
4.4	Sequence diagram	30
4.4.1	Message.....	30
4.4.1.1	To_out_TZ.....	30
4.4.1.2	To_in_TZ.....	30
4.5	State chart diagram	31
4.5.1	State.....	31
4.5.1.1	Trusted	31
4.5.1.2	Untrusted.....	31
5	GLOSSARY	32
6	REFERENCES.....	33



1 Introduction

1.1 Safety profile

This UML profile is made to insert several safety properties inside UML models. This insertion is used to map these properties from design to code.

Safety properties used in this profile are:

- reliability
- maintainability
- robustness

These properties are propagated through several abstraction levels to obtain code rule implementation. The modeller must identify each code rule with an identifier to use it inside UML diagrams (such as the class diagram).

In addition, this profile can be used to model fault propagation with sequence diagrams. The linking of failure and the safety state of the system can be modelled with state chart diagrams.

Finally, use case diagrams are used to model interactions with the system environment. Therefore, the modeller can integrate several aspects of the future system environment.

1.2 Security profile

This UML profile for security is made to map security properties with several abstraction levels. This mapping is usually used to certificate some systems, like with Common Criteria. This abstraction level mapping is based on a refinement of design system, and on the propagation of the security properties through each abstraction level.

Therefore, two types of view are concerned by this profile:

- Functional view : with use case diagram and state chart diagram
- Structural view : with component diagram and class diagram

The functional view supply items to model interactions which can occur between system and its environment. In this abstraction level, we describe potential attack by human actors or by other systems. In addition, through state chart diagrams, we can describe reactions of the system when it detects a potential attack. So it's possible to demonstrate that this system is always in a secure state.

The structural view contains several abstraction levels. To model them, a set of component diagram can be used with different structural details from high level design to the lowest level design. This level is usually represented by a class diagram. This



type of diagram provides a detailed description which can be used to verify the correct implementation of the security properties described in another abstract level. This verification is very important because the security properties must be propagated through all abstract levels in order to obtain a consistent security system.

Security properties used in this profile are:

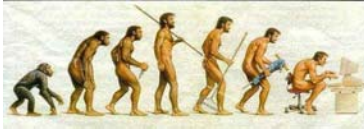
- Access control
- Data integrity
- Data secret

These properties are represented inside the security UML profile by stereotypes and tags. This representation doesn't provide any information about code structure. So we will not produce any code with this type of UML model.

This security UML profile doesn't provide any information about the manner to implement the security properties but provide the set of system elements which are included in security mechanism.

Tests are necessary to verify that the system is made in conformance with the security requirements.

Finally, the goal of this security UML profile is to provide a mapping of each security properties through all abstract levels. In addition, this profile can be used to do security evaluations in the scope of norms [CC].



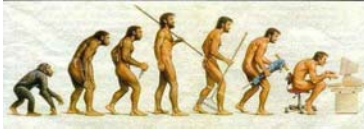
2 Notations

The following notations are used in this document:

<<xxxxxx>> : stereotype

{ xxxxxxx } : tag

[xxxxxxx] : tag value



3 Safety profile

3.1 Use case diagram

Impacted UML elements	Stereotypes	Tags
ACTOR	human	trusted
		untrusted
	not human	safe
		unsafe
USE CASE	critical	

3.1.1 Actor

Actor represents any physical entity which can interact with the system (other system, authorized use...). It is useful to distinguish actors by trusted level to model the future system environment.

Rule 3.1.1.a: Actor cannot be stereotyped with <<human>> and <<no human>> at the same time.

3.1.1.1 Human

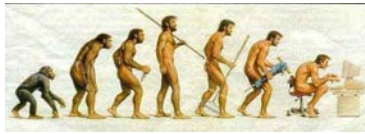
This stereotype enables the modeller to qualify an actor like a human entity. In this case, it must take into account its capabilities, potential mistakes and potential attacks. To do this, two tags are available: {trusted} and {untrusted}.

Rule 3.1.1.1.a: A human actor cannot be stereotyped with {trusted} and {untrusted} at the same time.

3.1.1.1.1 Trusted

A trusted actor is an authorized and trained system user. It is regarded as a trusted user and as a mistake source with a low occurrence probability. This tag must contain a trusted degree coming from a defined scale. For example, the scale can be an integer set. When this degree cannot be defined, the modeller can use [undefined].

Rule 3.1.1.1.1.a: {trusted} must contain a value (as integer) or [undefined].



3.1.1.1.2 Untrusted

A human actor is regarded as {untrusted} if it is not trained or authorized to use the system.

This tag represents the degree of nuisance capability of the untrusted user. A scale of nuisance capability must be made by the modeller. However, where this degree is undetermined, the modeller can use [undefined].

Rule 3.1.1.1.2.a: {untrusted} must contain a value (as integer) or [undefined].

3.1.1.2 *No human*

An actor stereotyped with <<no human>> represents all non human actors (such as system, software...). This type of actor must be split into safe actors (stereotyped with {safe}) and unsafe actors (stereotyped with {unsafe}).

Rule 3.1.1.2.a: A non human actor cannot be stereotyped with {safe} and {unsafe} at the same time.

3.1.1.2.1 Safe

A non human actor is regarded as {safe} if it complies with safety requirements or norms like [DO178B].

This tag must contain a safety degree coming from a defined scale. For example, the scale can be a set of integer. When this degree cannot be defined, the modeller can use [undefined].

Rule 3.1.1.2.1.a: {safe} must contain a value (as integer) or [undefined].

3.1.1.2.2 Unsafe

A non human actor is tagged with {unsafe} if it is not regarded as a safety actor. This tag represents the degree of tagged actor safety. If this degree is undetermined, the modeller can use [undefined].

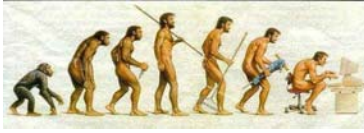
Rule 3.1.1.2.2.a: {unsafe} must contain a value or [undefined].

3.1.2 Use case

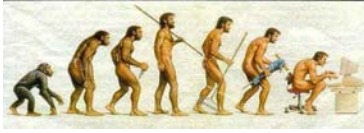
Use case is used as defined in [UML]. The use cases related to a critical functionality must be stereotyped with <<critical>>.

3.1.2.1 *Critical*

Use cases stereotyped with <<critical>> represent the use of a critical functionality by an actor. Criticality definition is the responsibility of the modeller.



Rule 3.1.2.1.a: A use case stereotyped with <<critical>> must be linked only with an actor stereotyped with <<human>> and tagged with {trusted} or with an actor stereotyped with <<no human>> and tagged with {safe}.



3.2 Component diagram

Impacted UML elements	Stereotypes	Tags
COMPONENT	reliability	
	maintainability	
	robustness	

3.2.1 Component

Components are used to describe the system to several levels of detail. A component is regarded as a consistent subset of more low level design elements. These elements are other components or, to the lowest level design, classes.

Several components can be compliant with safety requirements.

Rule 3.2.1.a: The <<reliability>>, <<maintainability>> and <<robustness>> stereotypes can be used on the same component at the same time.

3.2.1.1 Reliability

Components stereotyped with <<reliability>> must be reliable during their execution. In other words, they must maintain the service continuity of the global system.

3.2.1.2 Maintainability

Components stereotyped with <<maintainability>> must be designed to be simply modified by a developer. The evolution of these components is very important when the system must be transferred to another environment. Therefore, components which used to be in interface with the environment are particularly concerned by the <<maintainability>> stereotype.

3.2.1.3 Robustness

Components stereotyped with <<robustness>> must be implemented with the intention to be robust when the system is in an abnormal situation. Therefore, these components must minimize catastrophic event occurrence when the system runs near its limits.



3.3 Class diagram

Impacted UML elements	Stereotypes	Tags
CLASS	reliability	code rules
	maintenability	code rules
	robustness	code rules

3.3.1 Class

Classes are used as defined in [UML]. However, this profile adds a new principle: classes are refinements of component modelling in component diagrams. This refinement is made from the most abstract level to the most concrete level.

Rule 3.3.1.a: Each class must stem from only one component.

3.3.1.1 Reliability

A class stereotyped with <<reliability>> represents a code structure that must be compliant with reliability code rules. These rules are identified by the {code rules} tag.

Rule 3.3.1.1.a: Each class stemming from a component that is stereotyped with <<reliability>> must be stereotyped with <<reliability>>.

Rule 3.3.1.1.b: Each class stereotyped with <<reliability>> must be tagged with {code rules}.

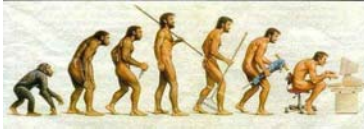
3.3.1.1.1 Code rules

This tag identifies code rules that must be implemented for the tagged class. These code rules are managed outside the model. They must be unambiguously identified to be used inside the model.

Rule 3.3.1.1.1.a: This tag must contain at least one value, whose format is free.

3.3.1.2 Maintainability

A class stereotyped with <<maintenability>> represents a code structure that must be compliant with reliability code rules. These rules are identified by the {code rules} tag.



Rule 3.3.1.2.a: Each class stemming from a component that is stereotyped with <<maintainability>> must be stereotyped with <<maintainability>>.

Rule 3.3.1.2.b: Each class stereotyped with <<maintainability>> must be tagged with {code rules}.

3.3.1.2.1 Code rules

This tag identifies code rules that must be implemented for the tagged class. These code rules are managed outside the model. They must be unambiguously identified to be used inside the model.

Rule 3.3.1.2.1.a: This tag must contain at least one value, whose format is free.

3.3.1.3 *Robustness*

A class stereotyped with <<robustness>> represents a code structure that must be compliant with reliability code rules. These rules are identified by the {code rules} tag.

Rule 3.3.1.3.a: Each class stemming from a component that is stereotyped with <<robustness>> must be stereotyped with <<robustness>>.

Rule 3.3.1.3.b: Each class stereotyped with <<robustness>> must be tagged with {code rules}.

3.3.1.3.1 Code rules

This tag identifies code rules that must be implemented for the tagged class. These code rules are managed outside the model. They must be unambiguously identified to be used inside the model.

Rule 3.3.1.3.1.a: This tag must contain at least one value, whose format is free.

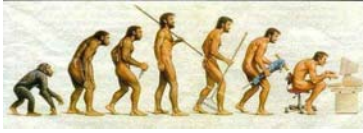


3.4 Sequence diagram

Impacted UML elements	Stereotypes	Tags
MESSAGE	fault	

3.4.1 Message

Messages are used as defined in [UML]. The <<fault>> stereotype is used to distinguish normal messages from abnormal messages. Abnormal messages indicate that the system is running in a degraded mode. Therefore, it is important for the modeller to monitor fault message propagation.



3.5 State chart diagram

Impacted UML elements	Stereotypes	Tags
STATE	reliable	
	error	
	failure	
TRANSITION	fault	

3.5.1 State

System states enable the modeller to distinguish normal running from abnormal (or degraded) running. Therefore, the modeller can describe one state stereotyped with <<reliable>>. This state describes the normal running of the system. To translate into another state, it must be a transition stereotyped with <<fault>>. These other states describe an abnormal running of the system, due to a fault. Two types of abnormal state exist:

- Failure state: stereotyped with <<failure>>
- Error state: stereotyped with <<error>>

Rule 3.5.1.a: A state cannot be stereotyped with <<reliable>> and with <<failure>> at the same time.

Rule 3.5.1.b: A state cannot be stereotyped with <<reliable>> and with <<error>> at the same time.

3.5.1.1 *Reliable*

A state stereotyped with <<reliable>> can be considered as a normal state of the system. In other words, during this state, the system provides the expected service.

3.5.1.2 *Error*

A state stereotyped with <<error>> characterizes the system after a fault occurrence, but before the moment when the provided service is degraded. Therefore, it is during this state that the propagation fault is carried out.



3.5.1.3 *Failure*

A state stereotyped with <<failure>> characterizes a degraded system running. This type of state appears after an occurrence and a propagation (or not) of a fault. In this state, the expected service is not provided.

Rule 3.5.1.3.a: A <<failure>> state can only appear after an <<error>> state.

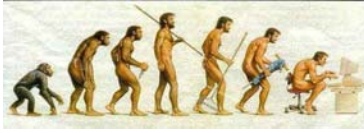
3.5.2 Transition

A transition represents an event that is the origin of system state changing. A transition stereotyped with <<fault>> characterizes the occurrence of a special event: a fault. In this case, the system switches into an <<error>> state.

3.5.2.1 *Fault*

A transition stereotyped with <<fault>> characterizes the occurrence of a fault causing a switch of state.

Rule 3.5.2.1.a: A transition stereotyped with <<fault>> must be a link from a state stereotyped with <<reliable>> to a state stereotyped with <<error>>.



4 Security profile

4.1 Use case diagram

Impacted UML elements	Stereotypes	Tags
ACTOR	trusted	roles
		privileges
	untrusted	roles
		privileges
	external	status
USE CASE	access control	roles
		privileges
	security	
ASSOCIATION	guarded access	conditions

4.1.1 Actor

Actor represents any physical entity which can interact with the system (other system, authorized user, attacker...).

Rule 4.1.1.a: Actor cannot be stereotyped with <<trusted>> and <<untrusted>> at the same time.

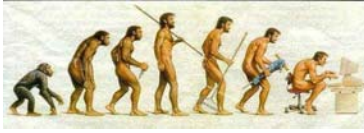
4.1.1.1 *Trusted*

An actor is <<trusted>> if he is considered trusted by another user who is trusted himself. A trusted actor is authorized to realize some use cases in accordance with his roles and privileges.

Rule 4.1.1.1.a: {roles} and {privileges} could be used at the same time.

Rule 4.1.1.1.b: At least, one of these tags must be used with <<trusted>>.

4.1.1.1.1 Roles



The notion of role is defined in [RBAC]. A role is a consistent set of privilege. This tag can take a set of value which represents the set of role assigned to the actor.

Rule 4.1.1.1.1.a: {roles} must take, at least, one value when it is used.

4.1.1.1.2 Privileges

A privilege enable authorized user to access to special functionalities (i.e. realize use cases). This tag lists the set of privilege assigned to an authorized user.

Rule 4.1.1.1.2.a: {privileges} must take, at least, one value when it is used.

4.1.1.2 *Untrusted*

An actor is <<untrusted>> if he is not considered trusted by another user who is trusted himself. But, this actor can realize some use cases in accordance with his privileges. He is considered like a potential attacker who has some privileges.

Rule 4.1.1.2.a: {roles} and {privileges} could be used at the same time.

Rule 4.1.1.2.b: At least, one of these tags must be used with <<untrusted>>.

4.1.1.2.1 Roles

The notion of role is defined in [RBAC]. A role is a consistent set of privilege. This tag can take a set of value which represents the set of role assigned to the actor.

Rule 4.1.1.2.1.a: {roles} must take, at least, one value when it is used.

4.1.1.2.2 Privileges

A privilege enable authorized user to access to special functionalities (i.e. realize use cases). This tag lists the set of privilege assigned to an authorized user.

Rule 4.1.1.2.2.a: {privileges} must take, at least, one value when it is used.

4.1.1.3 *External*

An actor is <<external >> if he is not authorized to use the system, but he could have some interactions with it. It is possible to assign him a status.

4.1.1.3.1 Status



{status} provide more information about the type of these actors. Three qualifiers can be used: [aggressive], [passive] or [unspecified].

The value [aggressive] corresponds to an actor who interacts in an aggressive and non-authorized manner with the system, with the intention to damage it. The value [passive] corresponds to a non-authorized actor (i.e. without privilege) who interacts with the system but without any malicious intention. [unspecified] value is used to describe actors who have undetermined intentions.

Rule 4.1.1.3.1.a: {status} tag can take only three values: [aggressive], [passive] or [unspecified].

4.1.2 Use case

A use case is used in the sense that is defined in UML 2.0. It corresponds to a functionality that can be used by an actor. The access to the use case (i.e. to the functionality) is limited by specific privileges (or specific roles). In addition, several use cases are considered as security elements because they enable actors to access to sensitive data or to modify security parameters.

Rule 4.1.2.a: A use case can be stereotyped with <<access control>> and with <<security>> at the same time.

Rule 4.1.2.b: A use case stereotyped <<security>> must be also stereotyped <<access control>>.

4.1.2.1 Access control

This stereotype enables to describe access restrictions to several use cases. These restrictions are made through privileges (or roles) necessary to realize several use cases.

Rule 4.1.2.1.a: {roles} and {privileges} could be used at the same time.

Rule 4.1.2.1.b: At least, one of these tags must be used with <<access control>>.

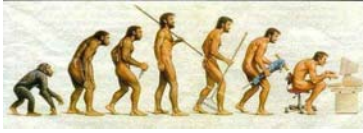
4.1.2.1.1 Roles

The notion of role is defined in [RBAC]. A role is a consistent set of privilege. This tag can take a set of value which represents the set of role required by the use case to realize it.

Rule 4.1.2.1.1.a: {roles} must take, at least, one value when it is used.

4.1.2.1.2 Privileges

This tag lists the set of privilege required by the use case to realize it.



Rule 4.1.2.1.2.a: {privileges} must take, at least, one value when it is used.

4.1.2.2 Security

This stereotype enables to specify if a use case belongs, or not, to security functionalities. These functionalities implement the configuration of security policies or their control.

The distinction between security functionalities and others is necessary in the frame of system certification, such as [CC].

In addition, access to these functionalities is critical. Therefore, developers must pay attention to their access control implementation.

Rule 4.1.2.2.a: A use case stereotyped with << security >> must be also stereotyped with << access control >>.

Rule 4.1.2.2.b: Only an actor stereotyped with << trusted >> can realize use case stereotyped with << security >>.

4.1.3 Association

Association is used in the sense defined in [UML]. It enables to specify that an actor can realize a use a case.

4.1.3.1 Guarded access

An association stereotyped with << guarded access >> enables to specify a restriction on the use case realization that is linked to the association. The privileges, that are owned by the actor, must be compliant with privilege requires by the use case. This compliance is specified by {condition}.

Rule 4.1.3.1.a: An association between an actor (stereotyped with << trusted >> or << untrusted >>) and a use case, stereotyped with << access control >>, must be stereotyped with << guarded access >>.

4.1.3.1.1 Condition

This tag must contain the condition to realize the use case. This condition is made with privileges (or roles) owned by the actor and required by the use case. This condition can be explained by an inclusion (strict or not) or an identity between these two sets of privilege (or role).

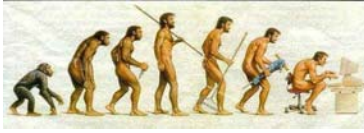
Example:

Ra: set of role owned by the actor
Ru: set of role required by the use case



{Condition: $R_a = R_u$ }

In this example, the set of role owned by the actor must be strictly equal to the set of role required by the use case.



4.2 Component diagram

Impacted UML elements	Stereotypes	Tags
COMPONENT	access control	roles
		privileges
INTERFACE	security	
	access control	roles
ASSOCIATION		privileges
	security	
	provided	
	required	
	guarded access	conditions

4.2.1 Component

The component is one of the features of UML 2.0. It is used to model a system in a static manner with several levels of detail. We use it such as a sub-system of the modelling global system. A sub-system is a consistent subset of low level design elements. These sub-systems can communicate themselves through several interfaces.

The modelling through several abstraction levels (from the most abstract level to the most concrete level) enables to map the concepts of the system with the implementation of this system. This mapping is required during an evaluation of the security system, such as [CC].

In this profile, we choose to propagate conditions and properties of access control.

Rule 4.2.1.a: A component, that provides an interface stereotyped with << security >>, must be stereotyped with << access control >>.

4.2.1.1 Access control

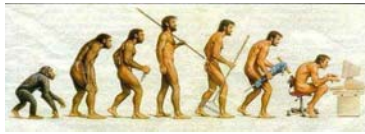
This stereotype specifies that the component owned privileges (or roles) to link it with another component.

Rule 4.2.1.1.a: {roles} and {privileges} could be used at the same time.

Rule 4.2.1.1.b: At least, one of these tags must be used with <<access control>>.

4.2.1.1.1 Roles

The notion of role is defined in [RBAC]. A role is a consistent set of privilege. This tag can take a set of value which represents the set of role assigned to the component to link itself with another component.



Rule 4.2.1.1.1.a: {roles} must take, at least, one value when it is used.

4.2.1.1.2 Privileges

This tag lists the set of privilege assigned to the component to link itself with another component.

Rule 4.2.1.1.2.a: {privileges} must take, at least, one value when it is used.

4.2.1.2 *Security*

This stereotype enables to specify if a component belongs, or not, to security functionalities. These functionalities implement the configuration of security policies or their control.

The distinction between security functionalities and others is necessary in the frame of system certification, such as [CC].

In addition, access to these functionalities is critical. Therefore, developers must pay attention to their access control implementation.

Rule 4.2.1.2.a: A component stereotyped with << security >> must be also stereotyped with << access control >>.

4.2.2 **Interface**

All interactions realized with a component must be done through an interface provided by the component. Interfaces must be represented inside the model.

A complete identification of interfaces is required by security evaluation norms such as [CC]. These interfaces, like components, own required roles and privileges to use it.

4.2.2.1 *Access control*

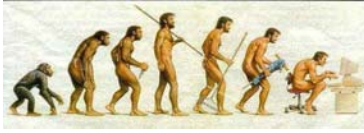
This stereotype is used to specify that the stereotyped interface using requires privileges (or roles).

Rule 4.2.2.1.a: {roles} and {privileges} could be used at the same time.

Rule 4.2.2.1.b: At least, one of these tags must be used with <<access control>>.

4.2.2.1.1 Roles

The notion of role is defined in [RBAC]. A role is a consistent set of privilege. This tag can take a set of value which represents the set of rols required to use the stereotyped interfaces.



Rule 4.2.2.1.1.a: {roles} must take, at least, one value when it is used.

4.2.2.1.2 Privileges

This tag lists the set of privilege required to use the stereotyped interface.

Rule 4.2.2.1.2.a: {privileges} must take, at least, one value when it is used.

4.2.2.2 *Security*

This stereotype enables to specify if an interface belongs, or not, to security functionalities. These functionalities implement the configuration of security policies or their control.

The distinction between security functionalities and others is necessary in the frame of system certification, such as [CC].

In addition, access to these functionalities is critical. Therefore, developers must pay attention to their access control implementation.

Rule 4.2.2.2.a: Each component stereotyped with << security >> must provide or use, at least, one interface stereotyped with <<security>>.

4.2.3 **Association**

An association is a link between a component and an interface. This link can specify two situations:

- the interface is provided by the component,
- the component use the interface which is provided by another component.

An association could link elements which own privileges to use it. In this case, association is realized only if its associated constraint is verified.

Rule 4.2.3.a: an association cannot be stereotyped, at the same time, << provided >> and << required >>.

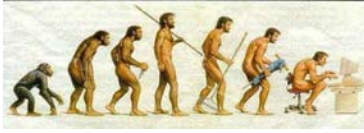
Rule 4.2.3.b: an association must link only one component with one interface.

4.2.3.1 *Provided*

This stereotype enables to specify that the component provides the interface with the intention to be accessible for other components.

4.2.3.2 *Required*

This stereotype enables to specify that the component uses the interface that is provided by another component.



4.2.3.3 *Guarded access*

An association stereotyped with << guarded access >> specifies a restriction to use or to provide an interface by a component. The privileges that are owned by the component must be compliant with privilege required to use or provide an interface. This compliance is specified by {condition}.

Rule 4.2.3.3.a: an association between a component that is stereotyped with << access control >> and an interface that is stereotyped with << access control >> must be stereotyped with << guarded access >>.

4.2.3.3.1 Condition

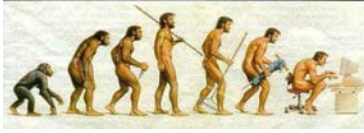
This tag must contain the condition to use or provide an interface. This condition is made with privileges (or roles) owned by the component and required by the interface. This condition can be explained by an inclusion (strict or not) or an identity between these two sets of privilege (or role).

Example:

Rc: set of role owned by the component
Ri: set of role required by the interface (to use or provide it)

{condition: Rc = Ri}

In this example, the set of role owned by the component must be strictly equal to the set of role required by the interface.



4.3 Class diagram

Impacted UML elements	Stereotypes	Tags
CLASS	secure	integrity
		secrecy
	interface	
	access control	role privilege
	security control	
ASSOCIATION	guarded access	condition
ASSOCIATION CLASS	security arbitrator	

4.3.1 Class

Classes are used like a final refinement of components (i.e. the most concrete representation) that are modelled in the previous diagrams. That's why we re-use same stereotypes and tags. This re-use enable to assure that the propagation of security properties is realized between each model level (i.e. from the most abstract level to the most concrete level).

In this type of diagram, more details can be specified with several class parameters.

In addition, new elements are appeared in these diagrams: association classes. They are used to specify more precisely how the association constraints must be implemented, in particular with the notion of "security arbitrator".

4.3.1.1 *Secure*

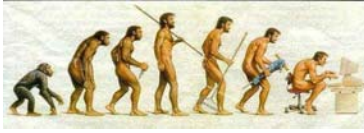
This stereotype enables to mark-up classes which own sensitive security parameters. Therefore these data must be protected in integrity (tagged with {integrity}) and/or in confidentiality (tagged with {secrecy}).

Rule 4.3.1.1.a: At least, one of the two tags must be used when the stereotype <<secure>> is used.

4.3.1.1.1 Integrity

This tag contains the list of parameters which must be protected in integrity manner.

4.3.1.1.2 Secrecy



This tag contains the list of parameters which must be protected in confidentiality manner.

4.3.1.2 Interface

This stereotype, already existed in UML norm, enable to model interface, described in more abstract level, such as a class (with parameters, methods...).

Rule 4.3.1.2.a: This stereotype must be used for classes which represent the interfaces specified in more abstract level.

4.3.1.3 Access control

This stereotype is used to specify the privileges (or roles) which are owned by the stereotyped class. But, if this stereotype is used with the stereotype << interface >>, it specifies the privileges required for the use of interface.

Rule 4.3.1.3.a: {roles} and {privileges} could be used at the same time.

Rule 4.3.1.3.b: At least, one of these tags must be used with <<access control >>.

4.3.1.3.1 Roles

The notion of role is defined in [RBAC]. A role is a consistent set of privilege. This tag can take a set of value which represents the set of role owned by the class. In case of this class is already stereotyped with << interface >> the set of value represents roles which are required to use this interface.

Rule 4.3.1.3.1.a: {roles} must take, at least, one value when it is used.

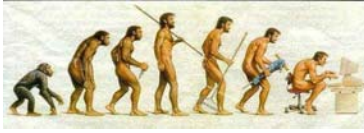
4.3.1.3.2 Privileges

This tag lists the set of privilege owned by the class or required to use the interface (i.e. class already stereotyped with << interface >>).

Rule 4.3.1.3.2.a: {privileges} must take, at least, one value when it is used.

4.3.1.4 Security control

Classes stereotyped with << security control >> are used to assure and control the system security. Therefore, the identification of these elements is necessary in the



frame of a system security evaluation. Several constraints must be taken during the code production to protect and test these elements. This identification is done through the use of <<security control>> stereotype.

4.3.2 Association

Association between classes enables to specify existing interactions between these classes. These associations are used in accordance with rules defined in [UML].

The meaning of interactions can change according to stereotyped classes impacted by these interactions.

4.3.2.1 *Guarded access*

An association stereotyped with << guarded access >> specifies a restriction to realize interactions between two classes or to use an interface by a class. The privileges, that are owned by the class, must be compliant with privilege requires to use an interface or to realize interactions with other classes. This compliance is specified by {condition}.

Rule 4.3.2.1.a: an association between a class that is stereotyped with << access control >> and << interface >> and a class that is stereotyped with << access control >> must be stereotyped with << guarded access >>.

4.3.2.1.1 Condition

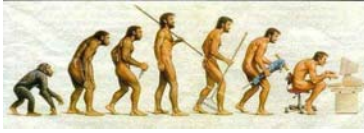
This tag must contain the condition to use an interface (i.e. class stereotyped with << interface >>) or realize an interaction with another class. This condition is made with privileges (or roles) owned by the class and required by the interface (i.e. class stereotyped with << interface >>) or by another class. This condition can be explained by an inclusion (strict or not) or an identity between these two sets of privileges (or roles).

4.3.3 Association class

4.3.3.1 *Security arbitrator*

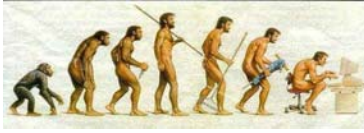
This element is used to specify classes which control or implement security policy. Their identification is required for security evaluation.

Association classes enable to model the mandatory side of several operations. Each association stereotyped with << guarded access >> can be strictly controlled by a class which managed security policy: class stereotyped with << security arbitrator >>.



SERIOUS
ITEA 04032
WP3 Deliverable 3.6 Final version
Page 29 of 33

Rule 4.3.3.1.a: Association class stereotyped with << security mediator >> must be also stereotyped with << security control >>.



4.4 Sequence diagram

Impacted UML elements	Stereotypes	Tags
Message	to_out_TZ	
	to_in_TZ	

4.4.1 Message

The stereotyped messages are used to differentiate that's go in to the trusted zone to that's go out from this zone. This differentiation is used to control sensitive data exchanges with another zone (trusted or not). In addition, it's used to make covert channel analysis.

Rule 4.4.1.a: A message cannot be stereotyped with << to_out_TZ >> and with << to_in_TZ >> at the same time.

4.4.1.1 To_out_TZ

Messages stereotyped with << to_out_TZ >> realize an output from the trusted zone. In this case, it's necessary to verify who send the message to assure data security.

Rule 4.4.1.1.a: Messages stereotyped with << to_out_TZ >> must be linked with an element inside the trusted zone to an element outside this zone.

4.4.1.2 To_in_TZ

Messages stereotyped with << to_in_TZ >> realize an input to the trusted zone. In this case, it's necessary to verify the innocuous of the message for sensitive data.

Rule 4.4.1.2.a: Messages stereotyped with << to_in_TZ >> must be linked with an element outside the trusted zone to an element inside this zone.



4.5 State chart diagram

Impacted UML elements	Stereotypes	Tags
State	trusted	
	untrusted	

4.5.1 State

The using of states to specify running system enables to anticipate a withdrawal solution for the system, in order to maintain its secure state.

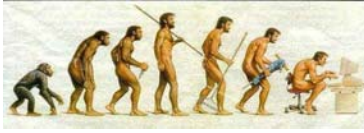
Rule 4.5.1.a: a state cannot be stereotyped with <<trusted>> and <<untrusted>> at the same time.

4.5.1.1 *Trusted*

A state of the system is << trusted >> if it enables the system to run normally and to assure data security.

4.5.1.2 *Untrusted*

A state of the system is << untrusted >> if it doesn't enable the system to run normally and to assure data security.



5 Glossary

Trusted zone: part of the system where sensitive data are protected, by a security policy, from attacks on their:

- Integrity
- confidentiality

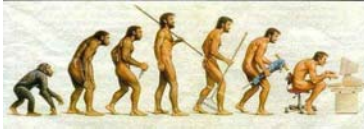
Secure state: A state which enables the system to:

- keep itself in good working status
- prevent information disclosure
- keep information integrity

RBAC: Role-Based Access Control

MLS: Multi-Level Security

TZ: Trusted Zone



6 References

[UML] OBJECT MANAGEMENT GROUP. *Unified Modelling Language version 2.0*. 2005

[DO178B] EUROCAE. DO-178B Software considerations in airborne systems and equipment certification. December 1992

[RBAC] R. SANDHU. *Advances in Computers. Role Base Access Control*. 1998

[CC] COMMON CRITERIA EDITORIAL BOARD. *Common Criteria for Information Technology Security Evaluation, version 2.2*. 2004