**SEVENTH FRAMEWORK PROGRAMME**
**Challenge 1**
**Information and Communication Technologies**

Trusted Architecture for Securely Shared Services

| | |
|---|---|
| **Document type:** | Deliverable |

| | |
|---|---|
| **Title:** | D4.3 – WP4 Implementation |

| | |
|---|---|
| **Work Package:** | WP4 |
| **Deliverable Number:** | D4.3 |
| **Editor:** | Dennis Vandevenne – K.U.Leuven |
| **Dissemination Level:** | Restricted |
| **Preparation Date:** | 31 May 2009 |
| **Version:** | 1.1 |



Information Society Technologies

**The TAS3 Consortium**

| Nr | Participant name | Country | Participant short name | Participant role |
|---|---|---|---|---|
| 1 | K.U.Leuven | BE | KUL | Coordinator |
| 2 | Synergetics nv/sa | BE | SYN | Project Manager |
| 3 | University of Kent | UK | KENT | Partner |
| 4 | University of Karlsruhe | DE | KARL | Partner |
| 5 | Technical University of Eindhoven | NL | TU/e | Partner |
| 6 | CNR/ISTI | IT | CNR | Partner |
| 7 | University of Koblenz-Landau | DE | UNIKOLD | Partner |
| 8 | Vrije Universiteit Brussel | BE | VUB | Partner |
| 9 | University of Zaragoza | ES | UNIZAR | Partner |
| 10 | University of Nottingham | UK | NOT | Partner |
| 11 | SAP research | DE | SAP | Partner |
| 12 | Eifel | FR | EIF | Partner |
| 13 | Intalio | FR | INT | Partner |
| 14 | Risaris | IR | RIS | Partner |
| 15 | Kenteq | BE | KETQ | Partner |
| 16 | Oracle | UK | ORACLE | Partner |
| 17 | Custodix | BE | CUS | Partner |

## Contributors

| | **Name** | **Organization** |
|---|---|---|
| 1 | Dennis Vandevenne | K.U.Leuven |
| 2 | Danny De Cock | K.U.Leuven |
| 3 | Marc Santos | UNIKOLD |
| 4 | Michele Bezzi | SAP |
| 5 | Jeroen Hoppenbrouwers | Synergetics/K.U.Leuven |

# Table of Contents



# Table of Figures

# 1 Executive Summary

The requirements assessment report (D1.2) has pointed out that *WP4 Information Protection* has two central high-level requirements. The first refers to the possibility to demonstrate to lay users the complex security and trust features of the TAS3 system. The second refers to the ability of providers to prove that they processed the information and services in accordance to the required policies.
The present version of this deliverable introduces a system independent and protocol agnostic prototype of the TAS3 ecosystem which addresses the first requirement. Future versions will extend this prototype to address the other WP4 requirements.
This prototype is built using the demonstrator framework of K.U.Leuven that provides an ecosystem within which different actor types can be defined and instantiated. These actors provide dummy functionality by default, but can easily be linked to real life instances of genuine service providers. Each of these actors (e.g., an information repository, a medical doctor, a hospital, an employment agency…) is able to communicate through the framework with each other actor. The security level of this communication (insecure, authenticating all outbound traffic, encrypting all communication, or authenticating and encrypting all communication), is specific to the policy of the actor, i.e., whether the actors can effectively communicate with one another is subject to both actors' respective policies. The demonstrator framework provides this communication policy enforcement by default.
The actors that have been specifically defined and instantiated for the TAS3 ecosystem (cf. section 5) are those necessary to illustrate the break-the-glass principle that has been elaborated on in D7. In addition to these actors, the TAS3 specific functionality has been developed and integrated in this framework.

***IMPORTANT CAVEAT**: The prototype presented in this deliverable focuses on showing the functionality and security properties of a TAS3 ecosystem with its actors using task driven business processes that currently operate on dummy actors, but that can easily be instantiated with genuine service providers and consumers. Because this deliverable is the outcome of the work of WP4 executed during the first year of a four year project, the presented implementation is not to be considered as a* reference *implementation in the strict sense: this prototype illustrates the functionality of the complex TAS3 ecosystem and is not intended to be used as a* reference *implementation to test against the functionality of a full blown TAS3 service provider or service consumer. The building blocks that implement the real-life components with standard-based protocols are being designed in work package 2, and will then be aligned with the work of the work packages 3, 5, 7, 8, 9. Subsequently, these building blocks will be integrated by work package 12. Testing the real-life components will be the object of work package 10.*

# 2 Introduction to the prototype

The following section will bring the reader to an understanding of how to install and execute the demonstrator framework (section 3), how to make use of the demonstrator framework (section 4), and how to create and run scenario's in general and the break the glass scenario in this particular case (section 5). Because this prototype is a work in progress, future releases are planned (see section 6 for the roadmap for future releases).

# 3 Available release and components

## 3.1 Software prerequisites

- Java runtime environment 1.6

## 3.2 Hardware prerequisites

- A genuine Belgian eID card or a developer's eID card can be used as smartcard based authentication method
- Note that userid/password authentication is supported to enable users who do not possess an eID card or developer's card

## 3.3 Installation Guidelines

1. Download the file http://www.esat.kuleuven.be/~decockd/tas3/wp4.prototype/TAS3_D4p3.wp4.implementation.20090526.tar.zip.
   This file includes all java class files, source files and external jar files that are necessary to successfully run and rebuild the prototype
2. Extract this file (unzip using the password `tasss' (without quotes), and untar the decrypted zip file). This will create a directory *shippy.20090526.*
3. Start the demonstrator framework:
   1. On Windows: execute *run.bat*
   2. On Unix: execute `. ./run.bat`

## 3.4 License information

The present prototype is licensed under the a triple license scheme, namely MPL 1.1/GPL 2.0/LGPL 2.1.

# 4 Tutorial

In this section the general use of the prototype is explained to enable users to easily create a scenario of their choosing. A scenario involves actors (service providers or service consumers) that live in their own context. Examples of actors that are active in a healthcare context are a Hospital, a Medical Doctor, a Patient, etc.
A typical scenario exists of a network of actors which belong to contexts. An actor can belong to one context when running one scenario, and to another context when running a different scenario. Therefore, first the creation of contexts and actors and the process of interconnecting them are explained. Then, the different views which the demonstrator framework offers to adjust and interact with a scenario are given and finally the saving and opening of scenarios is explained.
Adding functionality of a specific actor to the demonstrator framework involves straightforward Java-coding. The demonstrator framework has been programmed such that the number of lines and simplicity of this Java-coding can be kept to a minimum. The interaction of actors is managed through tasks.
Section 7 includes the source code of such an actor, its tasks and subtasks.
For simplicity reasons, the scenario that is used to illustrate the functionality of the presented prototype, is based on the break-the-glass scenario, cf. D7.1, and the consultation of summarized electronic health records (SumEHRs), cf. WP9.

## 4.1 Creating a scenario

Once the demonstrator framework is instantiated, the user is presented with an empty scenario set. This state can always be obtained by asking for a new scenario in the menu *file/new*. First, the scenario can be given a name with *scenario/rename scenario* and a background image with *scenario/add scenario background*. The scenario background image is positioned on the right top of the scenario view and is only decorative in nature. *File/save* allows the user to save the current scenario at any point (see section 4.7 Saving and Loading Scenarios).

## 4.2 Creating Contexts

After creating a new scenario, the necessary context(s) can be created with *context/add context* or the same functionality is available by right clicking the scenario background. The demonstrator enforces the uniqueness of each context name. After the creation of one or more contexts, more functionality becomes available in the context menu. Existing contexts can be renamed, given a particular color and the context visibility can be toggled.

- The renaming of a context enforces unique context names.
- To keep a clear view on the created scenario, it often is wise to give different contexts different colors.
- Toggling a context's visibility only makes sense when it contains one or more actors. This action changes the visibility of all actors in a particular context and of all edges which connect to at least one actor in the context.

More contexts can always be created and existing contexts can be adjusted when this seems necessary to the user.

## 4.3 Creating Actors

The creation of actors is available in the *edit/add actor* menu or in the pop-up menu which appears after a right click on the scenario background. The user then sees a target which

moves along with mouse movements. One needs to left-click to confirm the future location of the actor to be added or press escape to cancel the operation. After confirmation of the actor's location, the user is presented with a pop-up dialog which allows the quick creation of the necessary actor. This actor is chosen from the first drop down menu named *Choose actor type*. The second drop down box allows specifying the context in which the new actor will live. Furthermore, the user also needs to specify whether the actor will be a gateway[1] or not. The dialog which allows adding the actor (Figure 1) also contains a visibility path parameter and an object ID. The default values for these two parameters (as well as for the remaining fields) are sufficient for the creation of the break the glass scenario. Finally the user can confirm the creation of the actor, or choose to cancel the operation.

The present prototype illustrating the Break-the-Glass scenario involves the following actor types:

- Authoritative source of medical doctors: the authoritative source that is able to confirm whether a particular actor (in this case a general practitioner) is known as a medical doctor
- Authorization policy decision point: the PDP that will be involved in making the decision whether a service request (in this case fetching a patient's summarized eHealth record (SumEHR) from the patient's medical dossier)
- Crossroads bank of social security: a mediator service that abstracts the authoritative source of medical doctors from actors that are active in the social security context
- E-Health: a mediator service that makes the functionality of service providers in the healthcare context available to other service providers
- General practitioner: a medical doctor who manages the medical dossier of a patient
- General practitioners association: a grouping of two or more general practitioners. Each practitioner has access to the medical dossier of each patient of who is a client of the association of general practitioners
- Hospital: an actor that represents a hospital
- Medical doctor: a medical doctor stores the medical dossier of a patient in the database with medical dossiers of the general practitioners association to which he is associated
- Medical doctor trainee: a trainee of a medical doctor. A trainee does not yet have the necessary credentials that allows him/her to access a patient's SumEHR
- Patient: a patient is a client with a medical doctor
- Portal site: an actor that acts as an entry point for users
- SumEHR reference register: the authoritative source that specifies which association of general practitioners holds the medical dossier of a patient.

The present prototype also supports actors that are not relevant to illustrate the Break-the-Glass, but that can be used to illustrate other scenario's, e.g., registration of a newborn child, cross-context exchange of pseudonymized information, anonymization of information containers, aggregating financial and property information of a legal or natural person:

- Authoritative source of bailiffs: similar to the authoritative source of medical doctors, but dealing with bailiffs
- Bailiff: similar to a medical doctor, but providing bailiff functionality
- Belgian official journal: the authoritative source that is able to confirm whether a citizen is known to be a civil servant or not
- Cadaster: the authoritative source that is able to map land and houses to physical and legal persons
- Cross roads bank of enterprises: the authoritative source that hosts a database linking information on organizations and enterprises to their context specific identifiers

[1] An actor that is connected with another actor sees the functionality of the other actor. Actors are subdivided into two categories: gateways and non-gateways. A mediator service provider is a gateway that bridges the functionality of the actors that are connected with a gateway.

- Document anonymizer: a service provider that is able to anonymize and pseudonymize structured documents and information containers. This actor acts as the authoritative source of the pseudonymized information. These actors relate to the identifiers engine specified in D4.1
- Global identifier and document identifier anonymizer: actors that are responsible to map global identifiers (e.g., national IDs), document identifiers, etc. These actors relate to the identifiers engine specified in D4.1
- Federal, regional and local context identifier issuer: an actor that assumes the role of an identifier issuer active at a federal, regional and local level, respectively
- Federal service bus: an actor that acts a gateway to connect different actors that are active at the federal level
- Ministry of Finances: the authoritative source that hosts the financial information about natural and legal entities
- National Register: the authoritative source that hosts the reference information on natural persons
- Time stamping authority: a service provider that issues time stamps on information

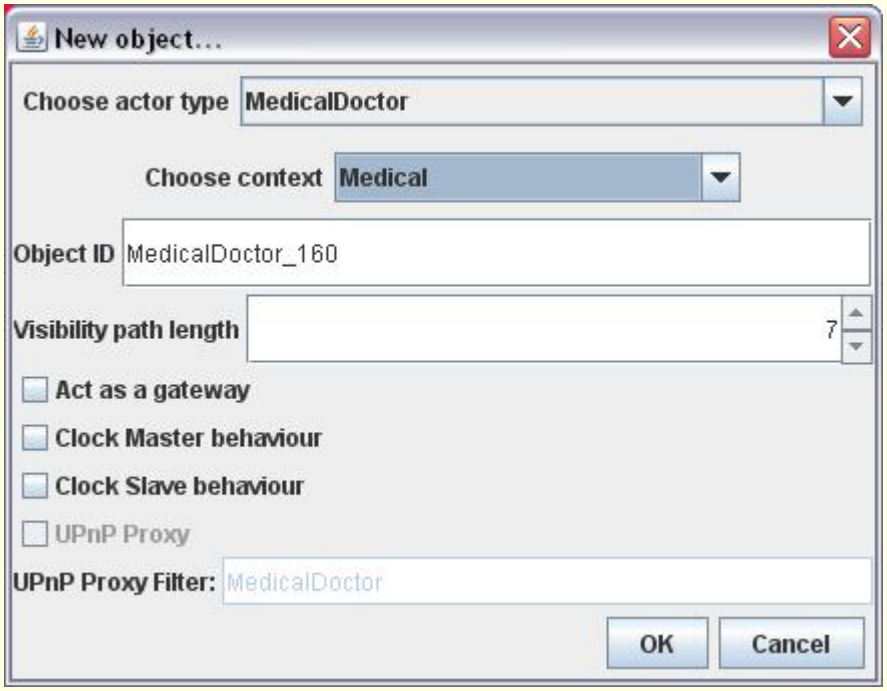


Figure 1: The user interface for adding an actor

## 4.4 Communication between Actors

For actors to be able to communicate in a task-driven business process, they need to be connected to each other. This can be done by connecting two actors directly (1) or by the design of a larger part of the scenario graph by which non-adjacent actors are able to communicate (2).

The first option is the simplest one. One actor needs to be right-clicked. Then, from the adaptive actor specific pop-up menu which appears, the *add edge* function can be selected. The scenario visualization will then draw the edge from the selected actor to the moving mouse pointer until a second actor is selected as edge end point or escape is pressed. For a scenario to be able to run correctly, all involved actors need to be connected as specified in the scenario description, see section 5.2.3.

The second option requires that the two actors which need to communicate are connected via one or more gateways, which can bridge communication messages and functionality offered

by actors. Whether or not actors are gateways is specified at the actor's creation, see section 4.3.

## 4.5 Different Views

The main view of the demonstrator is called *overview* (Figure 2: The overview of the break the glass scenario) and contains the current scenario graph which exists out of interconnected actors. Upon the selection of an actor its *inspector* is shown (Figure 2) which is a combination of a detailed view of the actor specific information and actor specific functionalities. The last view, at the bottom, is a tabbed pane which always contains the status window. This provides the user with feedback of the current operation of the demonstrator. Besides the status window, the bottom view will add a tab each time a user interface needs to be presented for the execution of a particular scenario. In this way multiple scenario components can run at the same time, without losing oversight on the offered functionality.

### 4.5.1 Scenario Graph Overview

This view, illustrated by Figure 2: The overview of the break the glass scenario, contains the current scenario graph which exists out of interconnected actors. It provides mechanisms for manipulating the scenario graph, the functionalities are:

- Creation of actors (see section 4.3), connections between actors (see section 4.4) and contexts (see section 4.2) by right clicking on the overview's background. This presents a menu with the necessary functions.
- The overview provides an interface for the selection of actors and edges. Any number of actors and edges can be selected by holding the *control-key* while selecting or by drawing a rectangle on the overview with the mouse *left button*.
- The positioning and repositioning of actors is supported. An actor is positioned by moving the presented location target visualization with the mouse while being in the process of adding an actor (see section 4.3). Actors can be repositioned by dragging the mouse or moving the arrow keys while one or more actors are selected. Holding the *shift-key* down while moving with the arrow-keys speeds up the movement. Edges cannot be moved, positioned or repositioned, because they connect already positioned actors and move along when their corresponding actors are repositioned.
- The addition and removal of actors is possible in the scenario graph overview. The addition of an actor is explained above. Deletion of one or more actors is possible by making a selection of the actor(s) to be removed and pressing the *delete key*.
- Advanced functionality is offered in the adaptive actor specific pop-up menu which is shown after a right click on any of the actors in the scenario graph overview. See below for more detailed explanation of these functionalities.
- Any number of selected actors can be quickly moved across contexts by using the numeric key pad. (Only supported for less than ten contexts)

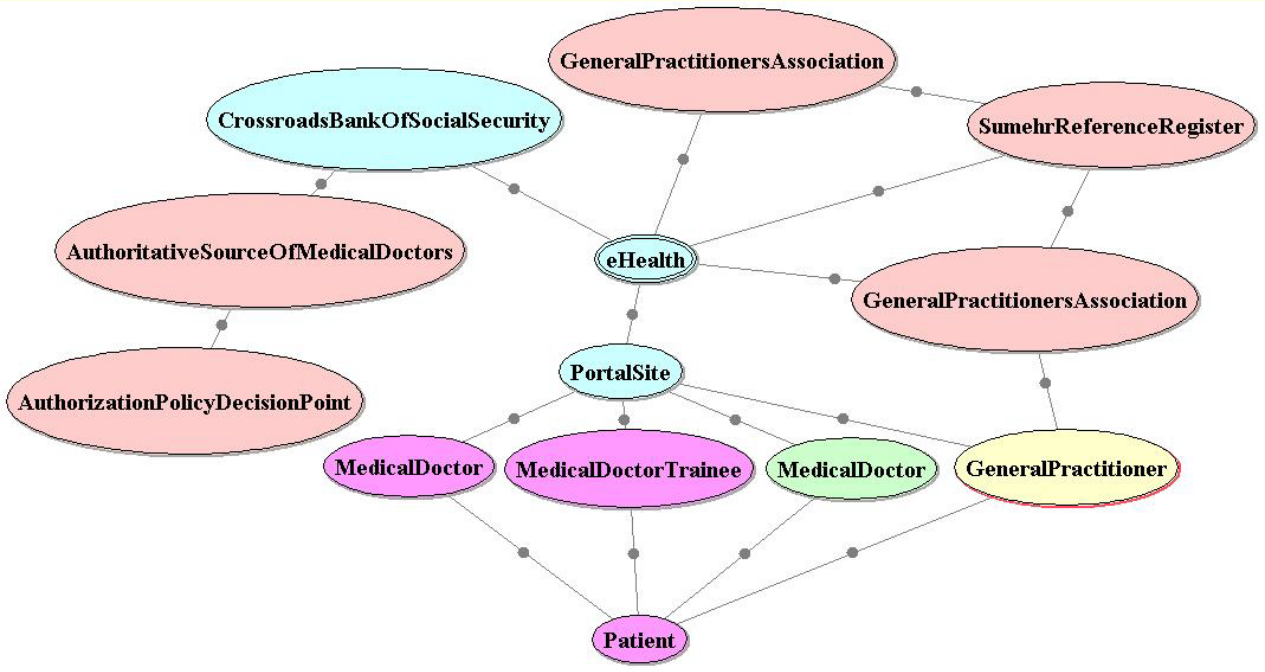


Figure 2: The overview of the break the glass scenario

Furthermore, the scenario graph overview offers actor specific functionalities by means of the actor specific, adaptive pop-up menu which appears by right-clicking on any of the actors in the scenario graph overview. The functionalities are the following:

- **Show inspector** shows the inspector of the selected actor. This inspector provides insight in the actor's events log. In a next version of this prototype, the inspector will consist of the Dashboard component that is specified in WP2, cf. D2.1, sections A.7 and D.3.
- **Add edge** fixes the starting point, the selected actor, of a new connection.
- **Add edge to all** makes a communication link between the selected actor and all other visible actors in the scenario graph overview.
- **Add edge to all in context** makes a communication link between the selected actor and all other visible actors in the context which can be selected.
- **Add icon** allows the user to browse for an icon to replace the default visualization of the actor. The chosen icon should be a representative visualization of the actor. To enable icons for actors, the *icon view* must be activated in the demonstrator's *view menu*.
- **Move to another context** offers a choice of contexts to which the selected actor can be moved to.
- **Only show this context** hides all actors of all other contexts than the context to which the selected actor belongs.
- **Only select this context** selects all actors of the context to which the selected actor belongs.
- **Rename context** asks the user to provide a new name for the context to which the selected actor belongs. Context name uniqueness is enforced by the demonstrator.
- **Change context color** provides an interface to select a new color for the context to which the selected actor belongs. All actors in that context will change color.
- **Clear task queue** clears all running tasks of the selected actor. This can be useful when the scenario is in an unwanted stated.
- **Show task queue** shows all running tasks of the selected actor in the actor's inspector (see section 4.5.2) which is added on the right of the scenario graph

overview. Although initially created for debugging purpose, this functionality could help the user to understand a running scenario.

- **Show log** opens a new browser window (currently only Firefox supported) in which the actor specific log is shown.
- **Visible actors** returns all actors which are visible to the selected actor. Directly connected actors are always visible. Other actors can be made visible by gateways.

### 4.5.2 Inspector view

An actor-specific inspector view can be obtained by right clicking any visible actor and choosing the *show inspector* option. If the *show inspector on selection* option is selected in the demonstrator's view menu, the selection of any actor directly leads to the visualization of its inspector.
Inspectors are grouped in a tabbed container on the right of the demonstrator (Figure 3). They have an informative function as well as an interactive function. The informative function provides transparency of the business processes that are executed while using the prototype. A business process is executed whenever a task or subtask is triggered. As stated before, the inspector view fulfills the role of the Dashboard that is specified in D2.1.
The second function supports auditing the active business process. All events that occur and an actor has to process are logged in an audit trail. The inspector provides access to these audit trails. The audit function is specified in D2.1, cf. section 6.3.

From top to bottom, an inspector's user interface provides the following components:

- **Inspector tabs** Any number of inspectors can be added to the inspector view. They will be available through the tabs on top, by which they can be closed as well. If the last actor specific inspector is closed, the inspector view will be hidden.
- **Visible objects tree** This is a tree-representation of the actors which are visible to the actor which inspector is used. Actors can be visible through direct connections, or they can be made visible through gateways. This is a legacy component which used to allow the invocation of services offered by other actors. The demonstrator now integrates scenario specific user interfaces at the bottom (see 4.5.3). The visible objects tree is still a useful component that allows overseeing the available services offered by the visible actors, although this functionality is also offered by the adaptive actor specific pop-up menu.
- **Actor's message trail** While running a scenario, e.g. the break the glass scenario, actors send and receive messages. This is shown in the actors' message trails.
- **Security toolbar** provides access to the actor’s security setting of the secure communications utility of the demonstrator framework. It enables point-to-point and end-to-end secure communications according to the security policy of the communicating actors. This security policy is managed by the actor itself and obliges the communicating party to send its replies using a security level that is at least equal to its own security level. This feature supports the user-centricity of specifying the security level at the communication layer. The term “secure” refers to the property that the information sent in a secure communications session (a so-called communications tube) is protected using one of the following security levels:
    - Not protected (**I**): the information is sent in an insecure way
    - Data-authenticity (**A**): the information is sent in such a way that the receiver can determine the sender of the information. The information is authenticated
    - Data-confidentiality (**C**): the information is sent so that only the intended sender and receiver have access to this information. The information is sent confidentially
    - Secure (**S**): the data-authenticity and data-confidentiality is protected at the same time. The sender of the information sent through a secure communications tube can indistinguishably be determined, and only the

intended sender and receiver have access to this information. The information is sent securely.

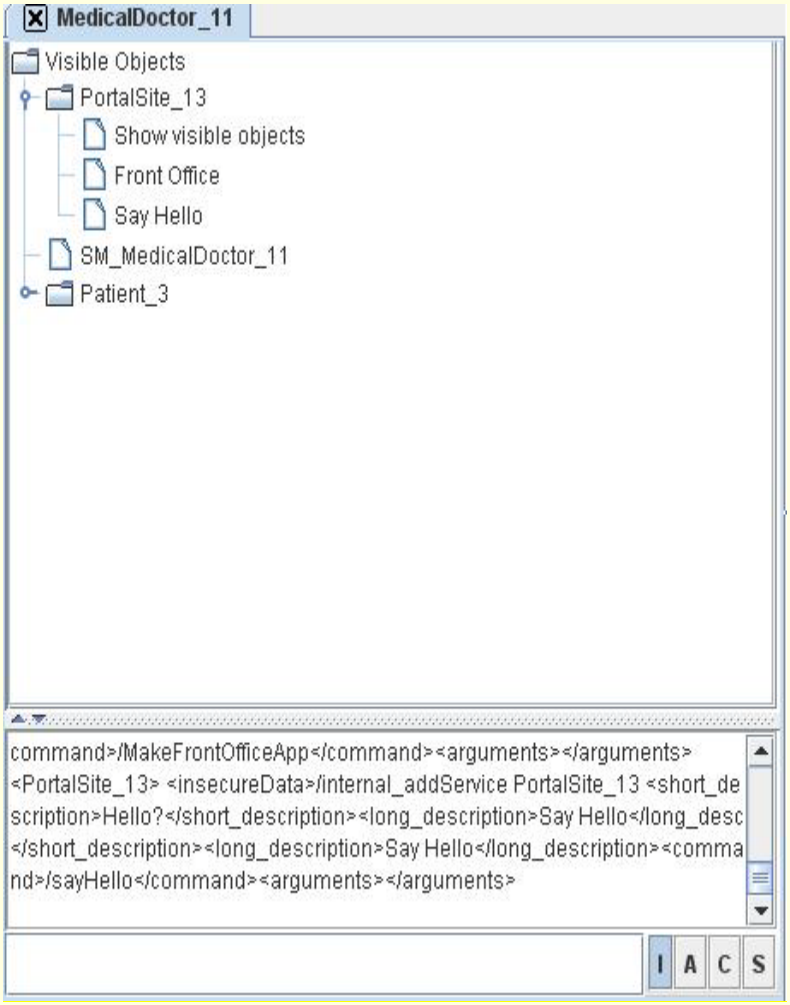


Figure 3: Inspector view of a Medical Doctor

### 4.5.3 Status and Application Front-end View

The last view, at the bottom, is a tabbed pane which always contains the status window. This provides the user with feedback of the current operation of the demonstrator. This feedback can be any general information, for instance, the status window could give a confirmation if a requested function, like saving a scenario (see section 4.7), is performed successfully. Furthermore, some details might be added dependent on the requested function, like for instance, the location where the scenario is saved.

Besides the status window, the bottom view will add a tab each time a user interface needs to be presented for the execution of a particular instantiation of the a scenario. In this way multiple scenario components can run at the same time, without loosing oversight on the offered functionality to the different actors.

## 4.6 Policy Manager

Every actor has a policy manager that manages XACML policies based on the actor's own functionality and the actors that are interconnected with this actor. The evaluation of these policies uses off-the-shelve PDPs that can easily be linked with those provided by WP5 and

WP7. The policy manager can be reached by selecting *Policy Manager* from the actor specific adaptive pop-up menu. When this is done the first time, a user interface with zero policies is presented. One can start adding one or more policies and later edit, send or remove them. The policy manager addresses the requirement with respect to the user-centricity of the policy enforcement: each user of an actor can specify which tasks can be used or have to be denied access to for any other know actor. The user can also specify which PDP should be responsible for evaluating the user's policy. This mechanism has been elaborated on in D7.1, section

- **Adding policies** is a functionality that allows the user to specify the policies he or she needs for the target scenario.
    - A selection needs to be made out of the available services the actor offers. At the writing of this document it is possible to create contradicting or overlapping policies for which the authorization PDP would return "indeterminate" as evaluation (see section 6).
    - The service requester needs to be specified from the drop down box that contains the actor's visible actors.
    - The result (permit or deny) of the policy needs to be specified.
    - The authorization policy decision point (authorization PDP) needs to be selected from the visible authorization policy decision points.
    - The current policy needs to be confirmed. This results in the policy summary.
- **Editing policies** is the functionality that allows existing policies to be changed. When an existing policy is changed, it needs to be resent to the selected authorization PDP.
- **Sending policies** can be done on a per policy base or for all policies at once. When a policy is edited, and needs to be resent, the user interface will make this clear by marking the *send* button red. It turns green after the authorization PDP confirms the receipt of the sent policy.
- **Removing policies** communicates the removal of the policy to the relevant authorization PDP and removes it from the policy collection of the actor's policy manager.
- **Saving the policy collection** to persistent storage is done when the policy manager is closed. In this way, policies are saved per actor and per newly created scenario.

Below (Figure 4) the user interface of the policy manager is shown. The first line represents a confirmed policy which is sent to the Authorization PDP. The second line represents a new policy being created. The authorization policy enforcement point (Authorization PEP) check determines whether the actor (the Authoritative Source for Medical Doctors in this case) will enforce the policies which are created by its policy manager user interface and sent to the authorization policy decision points.
The user has the possibility to specify any number of policies referring to any of the tasks supported by the actor. This illustrates that a user is able to specify fine grained policies in a convenient and user friendly manner.

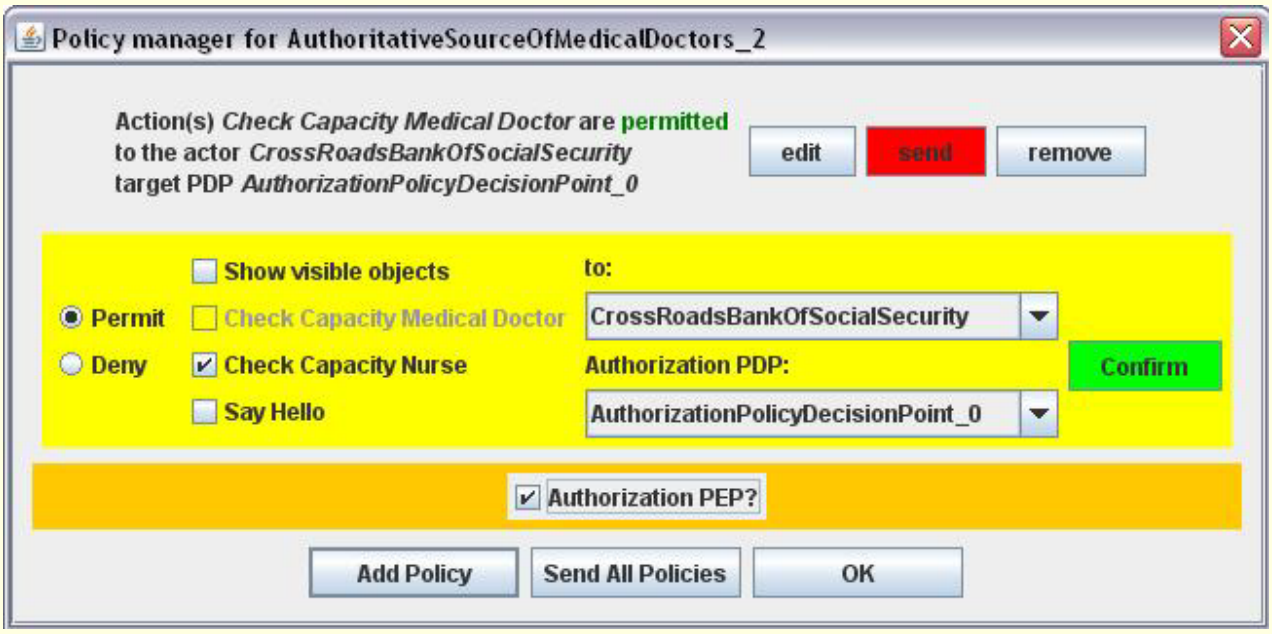


Figure 4: Policy manager user interface

## 4.7 Saving and Loading Scenarios

Created scenarios can be saved and loaded afterwards. Saving includes the scenario name, background, actors, interconnections between actors, status information about actors (for instance, being a gateway or actors' locations). Saving excludes remaining tasks in the actors' task queues, opened inspectors, opened scenario user interface instances. In general, running scenarios are not saved, the scenario network and its component are.

# 5 Running the software – Executing the Break the Glass Scenario

After giving a general explanation of the break the glass scenario and principle to consult a summarized electronic health record (also see work packages 7 and 9), this chapter walks the reader through both the break the glass scenario creation phase and the execution phase.

## 5.1 Description of the Break the Glass Scenario

In order to illustrate the break the glass scenario (cf. section 3 of D7.1), first two related scenarios are explained: Initialization of the relationship between the general practitioner (GP) and the patient (1) and a patient needing treatment in a hospital (2). Then the break the glass scenario illustrates a patient receiving treatment from a medical doctor in an emergency hospital. Finally, this last scenario is elaborated to treatment given by a medical doctor trainee, thus a delegation component added.

A general practitioners association (GPA) is a voluntary cooperation between general practitioners belonging to related medical disciplines. A general practitioner can prove his relationship to a general practitioners association by showing the voucher he receives from the general practitioners association.

There are three different types of patient files manipulated in the following scenarios: the patient's global medical dossier, electronic medical dossier and Summarized Electronic Health Record (SumEHR).

- The Global Medical Dossier (GMD) is kept by the patient's general practitioner or general practitioner's association. The GMD aggregates all medical information about the patient to avoid unnecessary double treatments and to provide a complete medical history when necessary.
- An Electronic Medical Dossier (EMD) is a file containing medical information about the patient which is stored by, for instance, a hospital where the patient receives treatment. The patient's GMD should refer to this EMD, accompanied with a summary of the treatment.
- The Summarized Electronic Health Record (SumEHR) contains useful medical information necessary for a specific treatment. Therefore, it contains pieces of medical information from the Global Medical Dossier, not the whole GMD. The SumEHR enables the communication between the patient's general practitioner of preference, and other medical professionals like, for instance, medical doctors working in a hospital.

### 5.1.1 Relationship initialization between the GP and the patient.

A patient can choose to initialize a global medical dossier (GMD) with a general practitioner of his or her choice. In the case that this general practitioner belongs to a general practitioners association, in fact, the patient chooses to initialize his global medical dossier with that general practitioners association. The global medical dossier is then stored by this general practitioners association. Furthermore, the general practitioners association needs to inform the SumEHR Reference Register of this new relationship between the patient and the GPA. This enables the SumEHR Reference Register to offer a service that refers to the correct GPA for a certain patient. The initialization described above is not a part of the break the glass scenario and is done before the scenario starts. The necessary data is stored in XML files related to the actors.

### 5.1.2 Patient needs treatment in a hospital

A patient feeling ill usually first visits his general practitioner, who diagnoses the patient and, if applicable, refers him or her to a medical hospital that offers the necessary treatment. The patient then goes to that treating hospital which initializes a patient file if none exists. This patient file contains a reference to the GPA of the patient's choice, as declared by the patient. After the patient has received treatment, the patient's file contains the relevant information about that treatment, for instance, a tissue analysis and the patient's general practitioner is notified of the results.
It is also possible to have a reversed information flow. When the patient revisits this general practitioner to discuss the results of the treatment, the general practitioner might need more information from the treating hospital besides the merely the results. Therefore, the general practitioner needs to request the patient file from the treating hospital which he can do at the eHealth portal site after successful authentication and authorization. In order for the treating hospital release the patient's file, the portal site application needs to include with the request the voucher the general practitioner received from his GPA which proves their relationship. The treating hospital then can verify whether the patient, granted access to his or her treatment data, belongs to this GPA by verifying the voucher and the local reference to the patients GPA.

### 5.1.3 Patient needs emergency treatment (Break the glass scenario)

A patient needing emergency treatment goes (or is brought) to the hospital closest to the patient's location. The emergency medical doctor then needs to be able to retrieve the SumEHR of the patient's GMD in order to be able to treat the patient.
To retrieve this information, the emergency MD turns to the eHealth portal site. He authenticates himself and requests a voucher to prove his capacity of medical doctor. In the back-office, eHealth mediates this service by forwarding the request to the Authoritative Source Medical Doctors (ASMD) which answers with a voucher that states the medical doctor's capacity and allows verifying this.
After successful authentication and authorization, the eHealth portal site offers the functionality to request a patient's SumEHR from the GMD stored at the GPA. EHealth, as mediator, first uses the service offered by the SumEHR Reference Register which returns a reference to the GPA it has learnt to associate to the patient's GMD. Then, the mediator can request a SumEHR from the referred GPA, which will refuse to release this information because the emergency medical doctor is not a member of the patient's GPA. The eHealth front-office notifies the emergency medical doctor and asks him whether he would like to "break the glass", which means that he can get access if he documents his request with a motivation and provides the necessary proof of capacity. The GPA will then return the requested SumEHR and log the emergency medical doctor's motivation and capacity voucher. This process is shown in Figure 5.
For this demonstration scenario, we assume this will result in enough medical information to treat the patient. In a real-life situation, it might be the case that this SumEHR refers to relevant medical information stored by medical practices or hospitals where the patient has had treatment before. Therefore it might be necessary to aggregate this information. This case is also included in Figure 5.

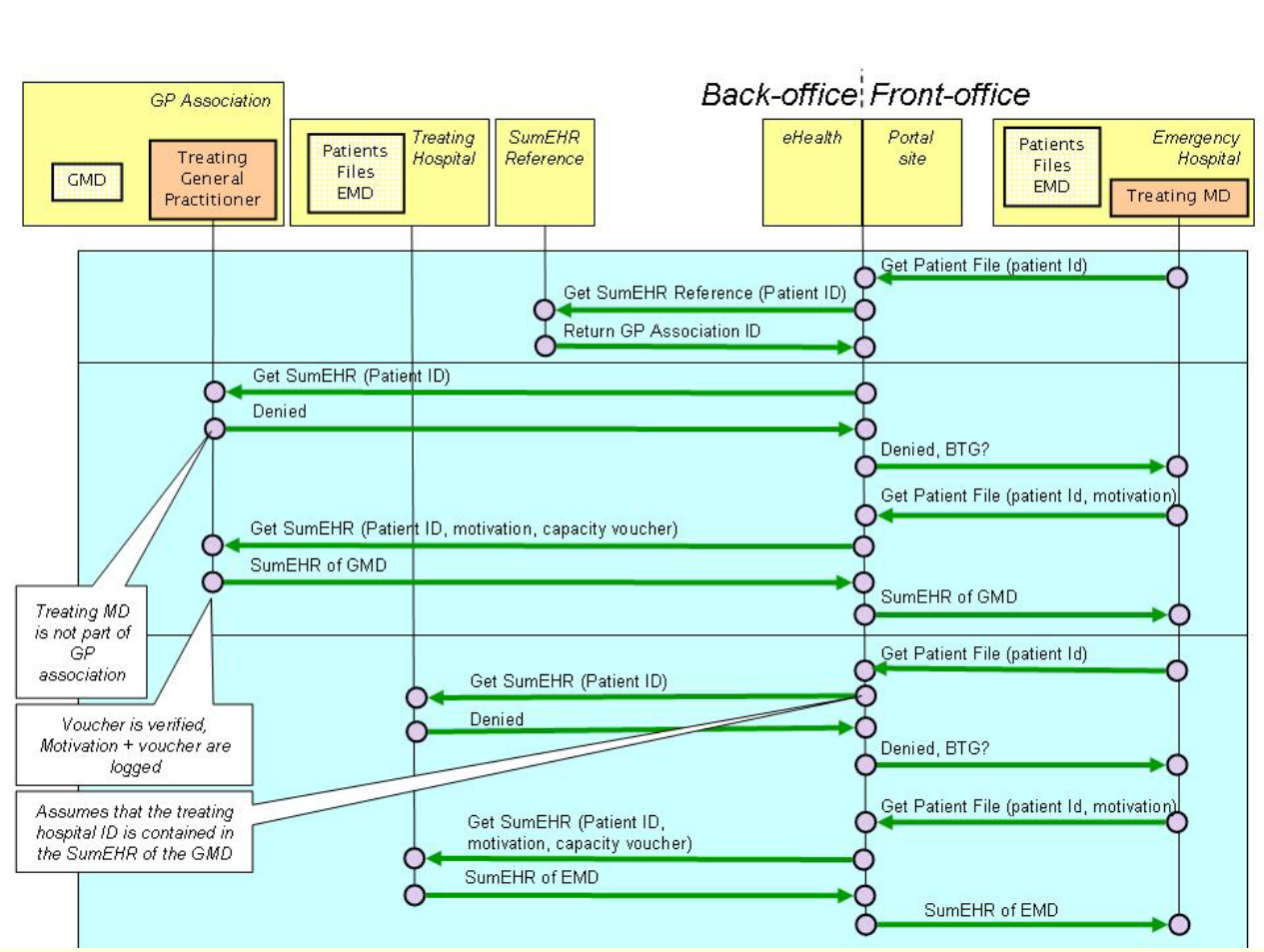


Figure 5: Break the glass scenario sequence diagram

### 5.1.4 Patient needs emergency treatment (Break the glass + delegation)

This scenario is similar to the one described above, except that a medical doctor trainee will attempt to break the glass. This will fail because his capacity voucher does not specify this capability. He will then request a delegation voucher from his supervising medical doctor. This will allow the trainee to break the glass.
This delegation component is not implemented in the current version of the scenario (see section 6).

## 5.2 Creating the break the glass scenario

First the different contexts are identified and created in section 5.2.1, then these contexts are populated with the, for the break the glass scenario required, actors in section 5.2.2. The third step, explained in section 5.2.3, entails interconnecting the actors where necessary. After this, the break the glass scenario is ready to be executed. This process is explained in section 5.3.

### 5.2.1 Creating the contexts

The process of creating a context in the demonstrator is explained in section 4.2. This process needs to be repeated for each context of the break the glass scenario. Although the

user is free to choose the context names (this will not break any functionality), we suggest the following for the break the glass scenario:

The (general) medical context
The federal context
The treating hospital context
The emergency hospital context
The general practitioner's practice context.

### 5.2.2 Creating the actors

The process of creating an actor in the demonstrator is explained in section 4.3. This process needs to be repeated for each actor of the break the glass scenario. The necessary information for the actors' creation is presented in Table 1. Note: at least two General Practitioners Associations need to be created.

| Actor | Context | Gateway |
|---|---|---|
| Medical Doctor | Emergency hospital | No |
| Medical Doctor Trainee | Emergency hospital | No |
| Medical Doctor | Treating hospital | No |
| General Practitioner | General practitioner's practice | No |
| Patient | Changing context:<br>General practitioner's practice<br>Treating<br>Emergency hospital | No |
| Authoritative Source Order Of Doctors | Medical | No |
| Authorization Policy Decision Point | Medical | No |
| SumEHR Reference Register | Medical | No |
| General Practitioners Association | Medical | No |
| Portal Site | Federal | No |
| Cross Roads Bank For Social Security | Federal | No |
| eHealth | Federal | Yes |

Table 1: Actors with their contexts and gateway status

### 5.2.3 Making the interconnections

The process of enabling actors to communicate with each other in the demonstrator is explained in section 4.4. This process needs to be repeated for each communication link between two actors of the break the glass scenario.

Connecting actors via an explicit link:

- Connect SumEHR Reference Register to all General Practitioners Associations in the medical context.
- Connect eHealth to all General Practitioners Associations.
- Connect eHealth to the SumEHR Reference Register and to the Portal Site.
- Connect the Cross Roads Bank For Social Security eHealth and to the Authoritative Source Order Of Doctors
- Connect the Authoritative Source Order Of Doctors to its authorization PDP.
- Connect the Portal Site to Medical Doctors, Medical Doctors Trainees and General Practitioners.
- Connect Patient to Medical Doctors, Medical Doctors Trainees and General Practitioners.
- Connect the General Practitioner to the General Practitioners association he or she belongs to. (Not necessary for the break the glass functionality, just for clarity of the actors' relations)

Connecting actors via gateway functionality forwarding:

- Connect the Cross Roads Bank for Social Security to the gateway eHealth.
- Connect the Flemish Portal Site to the gateway eHealth.

## 5.3 Executing the Break the Glass Scenario

To execute the break the glass scenario one right clicks the Medical Doctor actor from the emergency hospital context. From the list of visible actors (see section 4.5.1), the Federal Portal Site offers the *make front-office* functionality. By selecting this, the break the glass scenario user interface, also known as the front-office portal (see section 5.3.1), appears as a tabbed panel next to the status window, as explained in section 4.5.3 and shown in Figure 6.

### 5.3.1 The Front-Office

The front-office represents the user-interface towards the Medical Doctor actor. It allows for entity authentication, authorization and for requesting the patient file (with or without breaking the glass). It also gives feedback to the user about the status of these processes.

#### 5.3.1.1 Authentication

Entity authentication is implemented by a combination of token-based authentication and knowledge-based authentication. The Belgian eID card is chosen as the token, which is combined with the personal identification number (PIN). The Medical Doctor initiates the authentication process by clicking the *authenticate* button of the front-office user-interface. He will then be prompted for his personal identification number.
Users who do not have a Belgian eID card or a developer's card can user userid/password authentication. The default password is "pwd" (without quotes).
After successful authentication, the user obtains an authenticity voucher. The authentication process entails the communication of two actors, the Medical Doctor and the Portal Site.
In a real life situation, the authenticity voucher will have been issued by an IdP, as elaborated on in D2.1.

#### 5.3.1.2 Authorization

In order to be authorized to request a patient's SumEHR, the user needs to claim a capacity which entitles him or her to do so. In this example, the break the glass scenario, the user needs to claim the capacity of a Medical Doctor. This can be done with the Federal Portal front-office application where he can select this capacity and claim it with the authorize button.

The authorization process entails the communication of the Medical Doctor, the Federal Portal, the Cross Roads Bank for Social Security and the Authoritative Source Order of Doctors. The Federal Portal forwards the request to the Cross Roads Bank for Social Security, with eHealth in between as a gateway to forward the functionality. The Cross Roads Bank for Social Security has the information where to find an authoritative source which can verify the claimed capacity. In this case it is the Authoritative Source Order of Doctors which will do the verification and which will return proof, an authorization voucher, to the Cross Roads Bank for Social Security, which in its turn forwards the authorization voucher back to the service requestor. The identifier used across the different contexts is the national register number. Furthermore, the front-office also allows for multiple claimed capacities and the annulation of a single or all claimed capacities. Validating the authorization of an actor and the aggregation of the necessary attributes has been described in D7.1. Each actor involved in this authorization depends on the PDP to which it has been associated to determine the grant or deny decision.

#### 5.3.1.3 Requesting a patient's medical file

After the authentication and authorization step, the Federal Portal front-office user-interface allows the Medical Doctor to request the patient's medical file (Figure 6). The medical doctor needs to provide the patients ID (national registry number for this demonstrator). Then he or she can click on the *get patient SumEHR* button.

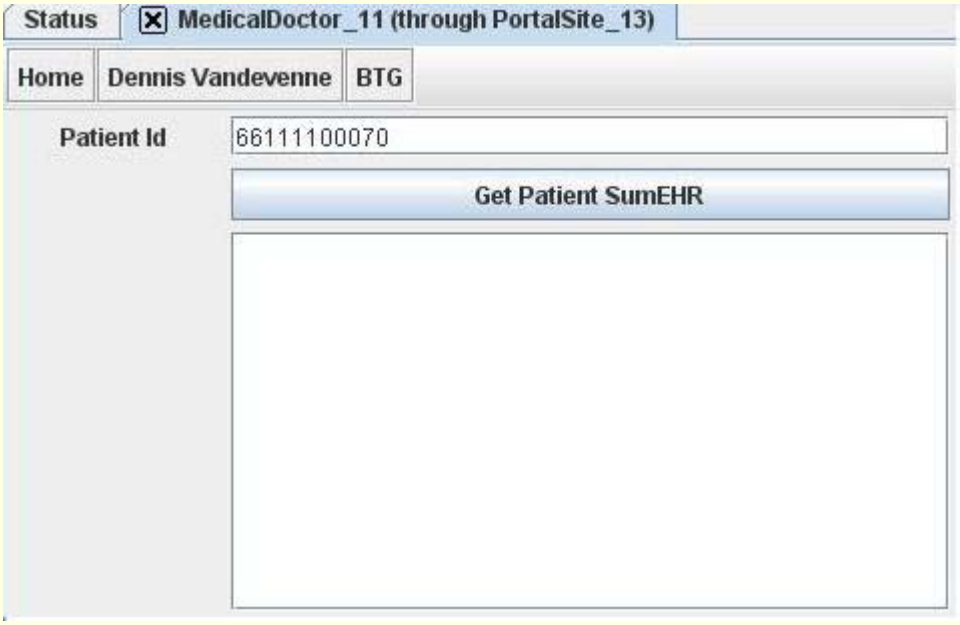


Figure 6: The Front-Office after initialization

If the medical doctor is a member of the General Practitioners Association, he will receive the requested patient's SumEHR. Otherwise, the user interface will prompt the medical doctor that he can only retrieve the patient's SumEHR by "breaking the glass". He can then check the "I want to break the glass" check box, which causes the necessary user interface elements to appear to enter the motivation, which is required to break the glass (Figure 7). A new attempt to retrieve the patient's SumEHR will then succeed and the General Practitioners Association will log the motivation provided by the Medical Doctor.

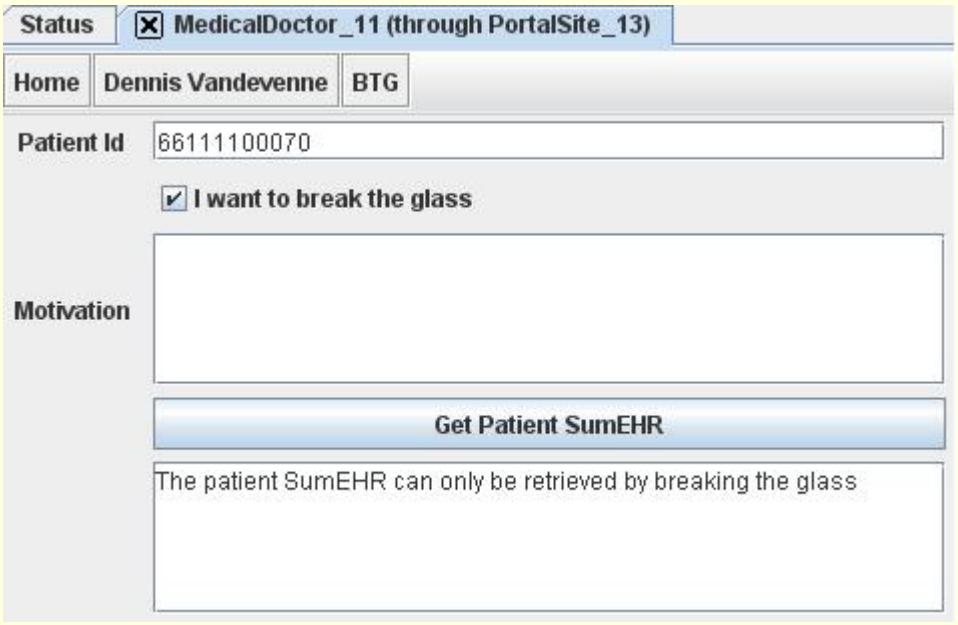


Figure 7: The user interface for breaking the glass

# 6 Roadmap for future release

| Version | Planned functionality addition |
|---|---|
| 1.01 | As delivered with this document |
| 1.1 | Version delivered to the Commission end of May 2009 |
| 1.2 | Policy Filter that ensures no contradicting or overlapping policies can be specified<br>GPA-MD vouchers should replace “medical doctors capacity vouchers + local DB check” |
| 1.3 | Redesign of the authentication and authorization user interface<br>Make policy persistence more intuitive in the Policy Manager |
| 1.4 | Add delegation component to the break the glass scenario (see section 5.1.4) in the form of a delegation manager |
| 1.5 | Alignment of the prototype with the use cases selected for the first pilot |
| 2.0 | Incorporate Trust and privacy policy negotiation |
| 2.1 | Version delivered to the Commission end of 2009 |
| | |

# 7 Examples of an actor's source code and task descriptions

## CrossRoadsBankOfSocialSecurity.java

```
/*****************************************************************************
 * ***** BEGIN LICENSE BLOCK * Version: MPL 1.1/GPL 2.0/LGPL 2.1
 *
 * The contents of this file are subject to the Mozilla Public License Version
 * 1.1 (the "License"); you may not use this file except in compliance with the
 * License. You may obtain a copy of the License at http://www.mozilla.org/MPL/
 *
 * Software distributed under the License is distributed on an "AS IS" basis,
 * WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for
 * the specific language governing rights and limitations under the License.
 *
 * The Initial Developer of the Original Code is Dennis Vandevenne.
 *
 * Portions created by the Initial Developer are Copyright (C) 2008 the Initial
 * Developer. All Rights Reserved.
 *
 * Contributor(s):
 *        Dennis Vandevenne <dennisvandevenne@gmail.com>
 *        Danny De Cock <godot@godot.be>
 *
 * Alternatively, the contents of this file may be used under the terms of
 * either the GNU General Public License Version 2 or later (the "GPL"), or
 * the GNU Lesser General Public License Version 2.1 or later (the "LGPL"),
 * in which case the provisions of the GPL or the LGPL are applicable
 * instead of those above. If you wish to allow use of your version of this
 * file only under the terms of either the GPL or the LGPL, and not to allow
 * others to use your version of this file under the terms of the MPL,
 * indicate your decision by deleting the provisions above and replace them
 * with the notice and other provisions required by the GPL or the LGPL. If
 * you do not delete the provisions above, a recipient may use your version
 * of this file under the terms of any one of the MPL, the GPL or the LGPL.
 *
 * ***** END LICENSE BLOCK *****
 */
package framework.administration.crossroadsbankofsocialsecurity.implementations;
import javax.swing.ImageIcon;

import framework.administration.AdministrationDefaults;
import framework.common.DefaultCommandsHandler;
import framework.common.FrameworkDefaults;
import framework.generic.genericobject.implementations.GenericEgovernmentObject;
import framework.generic.genericobject.implementations.GenericObjectCommandHandler;
import framework.generic.genericobject.implementations.taskmanaging.SubTask;
import framework.generic.genericobject.implementations.taskmanaging.Task;
import framework.generic.objectManager.implementations.ObjectManager;
import framework.policymanagement.utilities.PolicyManagementUtilities;
import framework.utilities.babel.TranslationEngine;
import framework.utilities.classnames.ClassPathUtilities;
import framework.utilities.egovernment.ArgumentParser;
public class CrossRoadsBankOfSocialSecurity extends GenericEgovernmentObject implements CrossRoadsBankOfSocialSecurityCommands,
                                                                                         CrossRoadsBankOfSocialSecurityPolicySettings {
    //
    public static final String        administrativeRoot               = AdministrationDefaults.administrativeRoot;
    public static final String        workingEnvironment               = AdministrationDefaults.workingEnvironment;
    //
    public final static Class         classCrossRoadsBankSocialSecurity  = CrossRoadsBankOfSocialSecurity.class;
    public static final String        fullClassName                    = ClassPathUtilities.
                                                                         fullClassNameOf(classCrossRoadsBankSocialSecurity);
    public final static String        capitalizedClassName             = ClassPathUtilities.
                                                                         classNameOf(classCrossRoadsBankSocialSecurity);
    //
    private final static String       objectFamily                     = AdministrationDefaults.objectFamily;
    //
    public final static String        frameTitle                       = capitalizedClassName;
    //
    public static boolean             useIcon                          = false;
    public final static String        objectImageIconFilename          = (useIcon)
                                                                           ? FrameworkDefaults.
                                                                         absoluteDirPathToIcon(objectFamily, capitalizedClassName)
                                                                           : noIcon;
    private final static java.net.URL imgURL                           = classCrossRoadsBankSocialSecurity.
                                                                         getResource(objectImageIconFilename);
    public final static ImageIcon     imageIcon                        = (imgURL != null) ? new ImageIcon(imgURL) : null;
    //
    private static final long         serialVersionUID                 = 0000000000000000001L;
    //
    private static final String       relativeClassPathToImplementations = FrameworkDefaults.
                                                                         relativeClassPathToImplementations(objectFamily, capitalizedClassName);
    //
    public static boolean             defaultActAsGateway              = false;
    public static boolean             defaultClockMaster               = false;
    public static boolean             defaultClockSlave                = false;
    //
    static {
        TranslationEngine.add(capitalizedClassName, TranslationEngine.ENGLISH, "CrossroadsBankOfSocialSecurity", "CBSS");
        TranslationEngine.add(capitalizedClassName, TranslationEngine.FRENCH, "BanqueCarrefourSecuriteSociale", "BCSS");
        TranslationEngine.add(capitalizedClassName, TranslationEngine.DUTCH, "KruispuntbankSocialeZekerheid", "KSZ");
    }
```

```
    //
    public CrossRoadsBankOfSocialSecurity(String objectId, Boolean actAsGateway, Integer visibilityPathLength, ObjectManager
            objectManager, Boolean clockMaster, Boolean clockSlave) {
        super(objectId, imageIcon, frameTitle, actAsGateway.booleanValue(), visibilityPathLength.intValue(), objectManager,
            clockMaster.booleanValue(), clockSlave.booleanValue(), false);
        //
        this.setObjectFamily(objectFamily);
        this.setFullClassName(fullClassName);
        PolicyManagementUtilities.initializePolicySettings(actionsThatAreAllowed, workingEnvironment, objectFamily,
           capitalizedClassName, myResource, relativeClassPathToImplementations, administrativeRoot, defaultActionsOfMine, this);
    }
    public void initCommands() {
        super.initCommands();
        this.addCommand(DECLARE_BIRTH_CMD, new GenericObjectCommandHandler(actionForMyResourceDeclareBirth) {
            public void handleCommand(String sourceObjectId, String arguments, String action) {
                getTaskMap().handleTask(sourceObjectId, getObjectId(), arguments, getActor());
            }
        });
        this.addCommand(CHECK_CAPACITY_MEDICAL_DOCTOR_CMD, new
                                        GenericObjectCommandHandler(actionForMyResourceCheckCapacityMedicalDoctor) {
            public void handleCommand(String sourceObjectId, String arguments, String action) {
                getTaskMap().handleTask(sourceObjectId, getObjectId(), arguments, getActor());
            }
        });
        this.addCommand(CHECK_CAPACITY_NURSE_CMD, new GenericObjectCommandHandler(actionForMyResourceCheckCapacityNurse) {
            public void handleCommand(String sourceObjectId, String arguments, String action) {
                getTaskMap().handleTask(sourceObjectId, getObjectId(), arguments, getActor());
            }
        });
        this.addCommand(CHECK_CAPACITY_CIVIL_SERVANT_CMD, new
                                         GenericObjectCommandHandler(actionForMyResourceCheckCapacityCivilServant) {
            public void handleCommand(String sourceObjectId, String arguments, String action) {
                getTaskMap().handleTask(sourceObjectId, getObjectId(), arguments, getActor());
            }
        });
        this.addCommand(CHECK_CAPACITY_BAILLIF_CMD, new GenericObjectCommandHandler(actionForMyResourceCheckCapacityBaillif) {
            public void handleCommand(String sourceObjectId, String arguments, String action) {
                getTaskMap().handleTask(sourceObjectId, getObjectId(), arguments, getActor());
            }
        });
        this.addCommand(REQUEST_TO_KSZ_CMD, new GenericObjectCommandHandler(actionForMyResourceRequestToKsz) {
            public void handleCommand(String sourceObjectId, String arguments, String action) {
                if(ArgumentParser.getArgumentFromString(arguments, "id").equals("requestToKSZ")) {
                    Task currentTask = taskMap.handleEndTask(sourceObjectId, getObjectId(), arguments, getActor());
                    String result = ArgumentParser.encapsulateInTags("answer by KSZ", "answer");
                    result += ArgumentParser.encapsulateInTags(ArgumentParser.getArgumentFromString(arguments, "servicerequester"),
                       "servicerequester");
                    taskMap.handleTaskDone(currentTask, result);
                }
            }
        });
        this.addCommand(TASK_DONE_CMD, new GenericObjectCommandHandler(actionForMyResourceTaskDone) {
            public void handleCommand(String sourceObjectId, String arguments, String action) {
                taskMap.handleSubTaskDone(arguments); // handle one of the subtasks
                String doneSubTaskSeqNr = ArgumentParser.getArgumentFromString(arguments, "seqnr");
                SubTask currentSubTask = taskMap.findSubtask(Integer.valueOf(doneSubTaskSeqNr).intValue());
                if(currentSubTask == null) {
                    return;
                }
                if(ArgumentParser.getArgumentFromString(arguments, "birthDeclarationStatus") != null) {
                    if(currentSubTask.getTaskID().equalsIgnoreCase("declareBirth")) {
                        SubTask currentst = currentSubTask.getParrentTask().getSubtask("getRegionalContextIdentifier");
                        if(Boolean.parseBoolean(ArgumentParser.getArgumentFromString(currentSubTask.getResult(), "accepted"))) {
                            if(currentst != null) {
                                if(currentst.getStatus() == 0) { //if idle, of course it's idle, "paranoid check"
                                    currentst.send();
                                    getActor().getTaskMap().show(); //debug
                                }
                            }
                        }
                        else {
                            System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
                            System.out.println("eBirth senario terminated because birth declaration was rejected\n" +
                                "no data is fed to the regional and local contexts\n" + "birth needs to be declared again");
                            System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
                            getTaskMap().removeTask(currentst.getParrentTask());
                        }
                    }
                }
                else if(ArgumentParser.getArgumentFromString(arguments, "regionalcontextidentifierdaddy") != null) {
                    if(currentSubTask.getTaskID().equalsIgnoreCase("getRegionalContextIdentifier")) {
                        SubTask currentst = currentSubTask.getParrentTask().getSubtask("feedDataToCorve");
                        // TODO: implement in generic way
                        currentst.setReceivedInputArguments(currentst.getReceivedInputArguments() +
                                ArgumentParser.encapsulateInTags("some data", "dataForCorve"));
                        if(currentst != null) {
                            if(currentst.getStatus() == 0) { //if idle, of course it's idle, "paranoid check"
                                currentst.send();
                                getActor().getTaskMap().show(); //debug
                            }
                        }
                    }
                }
            }
        });
        DefaultCommandsHandler.addDefaultCommands(this);
    }
}
```

## CrossRoadsBankOfSocialSecurityTasks/tasks.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="tasks.xsl"?>

<tasks>
	<actor>CrossRoadsBankOfSocialSecurity</actor>

	<task>
		<id>declareBirth</id>
		<requiredArguments>
			<requiredArgument>declarerRRN</requiredArgument>
			<requiredArgument>childName</requiredArgument>
			<requiredArgument>childSurName</requiredArgument>
			<requiredArgument>rrndaddy</requiredArgument>
			<requiredArgument>rrnmommy</requiredArgument>
			<requiredArgument>servicerequester</requiredArgument>
			<requiredArgument>authenticityvoucher</requiredArgument>
			<requiredArgument>capacityvoucher</requiredArgument>
		</requiredArguments>
		<requiredResults>
			<requiredResult>birthDeclarationStatus</requiredResult>
			<requiredResult>rrnbaby</requiredResult>
			<requiredResult>servicerequester</requiredResult>
		</requiredResults>
		<subtasks>
			<subtask>
				<id>declareBirth</id>
			</subtask>
			<subtask>
				<id>getRegionalContextIdentifier</id>
			</subtask>
			<subtask>
				<id>feedDataToCorve</id>
			</subtask>
		</subtasks>
	</task>

	<task>
		<id>requestToKSZ</id>
		<requiredArguments>
			<requiredArgument>personalIdentifier</requiredArgument>
			<requiredArgument>organizationIdentifier</requiredArgument>
			<requiredArgument>servicerequester</requiredArgument>
		</requiredArguments>
		<requiredResults>
			<requiredResult>answer</requiredResult>
			<requiredResult>servicerequester</requiredResult>
		</requiredResults>
		<subtasks>
			<subtask>
				<id>requestToKSZ</id>
			</subtask>
		</subtasks>
	</task>

	<task>
		<id>CheckCapacityMedicalDoctor</id>
		<requiredArguments>
			<requiredArgument>personalIdentifier</requiredArgument>
			<requiredArgument>servicerequester</requiredArgument>
		</requiredArguments>
		<requiredResults>
			<requiredResult>capacityvoucher</requiredResult>
			<requiredResult>servicerequester</requiredResult>
```

```
            </requiredResults>
            <subtasks>
                  <subtask>
                        <id>CheckCapacityMedicalDoctor</id>
                  </subtask>
            </subtasks>
      </task>

      <task>
            <id>CheckCapacityNurse</id>
            <requiredArguments>
                  <requiredArgument>personalIdentifier</requiredArgument>
                  <requiredArgument>servicerequester</requiredArgument>
            </requiredArguments>
            <requiredResults>
                  <requiredResult>capacityvoucher</requiredResult>
                  <requiredResult>servicerequester</requiredResult>
            </requiredResults>
            <subtasks>
                  <subtask>
                        <id>CheckCapacityNurse</id>
                  </subtask>
            </subtasks>
      </task>

      <task>
            <id>CheckCapacityCivilServant</id>
            <requiredArguments>
                  <requiredArgument>personalIdentifier</requiredArgument>
                  <requiredArgument>servicerequester</requiredArgument>
            </requiredArguments>
            <requiredResults>
                  <requiredResult>capacityvoucher</requiredResult>
                  <requiredResult>servicerequester</requiredResult>
            </requiredResults>
            <subtasks>
                  <subtask>
                        <id>CheckCapacityCivilServant</id>
                  </subtask>
            </subtasks>
      </task>

      <task>
            <id>CheckCapacityBailiff</id>
            <requiredArguments>
                  <requiredArgument>personalIdentifier</requiredArgument>
                  <requiredArgument>servicerequester</requiredArgument>
            </requiredArguments>
            <requiredResults>
                  <requiredResult>capacityvoucher</requiredResult>
                  <requiredResult>servicerequester</requiredResult>
            </requiredResults>
            <subtasks>
                  <subtask>
                        <id>CheckCapacityBailiff</id>
                  </subtask>
            </subtasks>
      </task>
</tasks>
```

### CrossRoadsBankOfSocialSecurityTasks/subTasks/declareBirth.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="subtask.xsl"?>

<subtask type="async">
	<id>declareBirth</id>
	<target>RijksregisterRegistreNational</target>
	<requiredArguments>
		<requiredArgument>declarerRRN</requiredArgument>
		<requiredArgument>childName</requiredArgument>
		<requiredArgument>childSurName</requiredArgument>
		<requiredArgument>rrndaddy</requiredArgument>
		<requiredArgument>rrnmommy</requiredArgument>
		<requiredArgument>servicerequester</requiredArgument>
		<requiredArgument>authenticityvoucher</requiredArgument>
		<requiredArgument>capacityvoucher</requiredArgument>
	</requiredArguments>
	<requiredResults>
		<requiredResult>birthDeclarationStatus</requiredResult>
		<requiredResult>rrnbaby</requiredResult>
		<requiredResult>servicerequester</requiredResult>
	</requiredResults>
	<timeout>10000</timeout>
</subtask>
```

### CrossRoadsBankOfSocialSecurityTasks/subTasks/getLocalContextIdentifier.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="subtask.xsl"?>

<subtask>
	<id>getLocalContextIdentifier</id>
	<target>LocalContextIdentifierIssuer</target>
	<requiredArguments>
		<requiredArgument>daddyRRN</requiredArgument>
	</requiredArguments>
	<requiredResults>
		<requiredResult>localcontextidentifier</requiredResult>
	</requiredResults>
	<timeout>600000</timeout>
</subtask>
```

# 8 Document Control

## Amendment History

| Version | Baseline | Date | Author | Description/Comments |
|---|---|---|---|---|
| 1.0 | | 15/04/2009 | DVDV | First draft, description of the demonstrator framework V.0. |
| 1.01 | | 19/04/2009 | DDC | Submitted to review team |
| 1.02 | | 29/05/2009 | DDC | Final version |