FP6-2004-27020

Access-eGov

Access to e-Government Services Employing Semantic Technologies

Instrument: STREP

Thematic Priority:
SO 2.4.13 Strengthening the integration
of the ICT research effort in an enlarged Europe

D3.2 Access-eGov Components Functional Descriptions

Start date of project: January 1, 2006 **Duration:** 36 months

Date of submission: March 30, 2007

Lead contractor for this deliverable: InterSoft, a. s.

Revision: Final 2.0

Dissemination level: PU

Acknowledgement: The Project is funded by European Commission DG INFSO under the IST programme, contract No. FP6-2004-27020.

FP6-2004-27020 **Access-eGov**

Access to e-Government Services Employing Semantic Technologies

D3.2 Access-eGov Components Functional Descriptions

Workpackage:	WP3	**Task:**	T3.3
Date of submission:	March 30, 2007		
Lead contractor for this deliverable:	InterSoft, a. s.		
Authors:	Martin Tomášek (IS) Marek Paralič (IS) Karol Furdík (IS) Lukasz Ryfa (EMA) Dominik Smogór (EMA) Andrzej Marciniak (EMA) Stefan Dürbeck (UR) Rolf Schillinger (UR) Marek Skokan (TUK) Peter Bednár (TUK) Ján Hreňo (TUK) Tomáš Sabol (TUK)		
Version:	2.0		
Revision	Final		
Dissemination level:	PU		

Project partners:

Technical University of Kosice (TUK), Slovakia (Coordinator); University of Regensburg (UR), Germany; German University in Cairo (GUC, Egypt; InterSoft, a. s. (IS), Slovakia; EMAX S.A. (EMA), Poland; Kosice Self-Governing Region (KSR), Slovakia; Cities on Internet Association (COI), Poland; e-ISOTIS (ISO), Greece; Municipality of Michalovce (MI), Kosice; City Hall of Gliwice (GLI), Poland; State Government of Schleswig-Holstein (SHG), Germany.

Abstract:

This document describes in detail the list of software modules and components that will comprise the architecture in Access-eGov project. For each module we provide the definition, the function it performs inside the overall architecture, the components needed to implement the module, the relation between those components and their communication.

Document Sign-off

Nature of Sign-off (Reviewed/Approved/Submitted)	Name	Role	Participant short name	Date
Reviewed	Martin Tomášek	TL	IS	Mar 30, 2007
Approved	Martin Tomášek	WPL	IS	Mar 30, 2007
Submitted	Tomáš Sabol	PM	TUK	Mar 30, 2007

Document Change Record

Date	Version	Contributor(s)	Change Details
Oct 4, 2006	0.1	Marek Paralič (IS)	First draft
	0.2	Lukasz Ryfa (EMA)	Chapter 10
Oct 20, 2006	0.3	Martin Tomášek (IS)	Chapter 8
Oct 20, 2006	0.4	Stefan Dürbeck (UR)	Chapter 6
Oct 31, 2006	0.5	Lukasz Ryfa (EMA)	Chapter 10 reviewed
Nov 1, 2006	0.6	Martin Tomášek (IS)	Revision after meeting at UR
Nov 7, 2006	0.7	Dominik Smogór (EMA)	Chapter 10
Nov 9, 2006	0.8	Stefan Dürbeck (UR)	Chapter 6
Nov 10, 2006	0.9	Martin Tomášek (IS)	Revision and merging of previous versions
Nov 10, 2006	0.10	Marek Skokan, Peter Bednár (TUK)	Chapters 5, 7, 8. Revision
Nov 10, 2006	0.11	Martin Tomášek (IS), Tomáš Sabol (TUK), Peter Bednár (TUK), Dominik Smogór (EMA)	Revision
Nov 10, 2006	1.0	Marek Skokan (TUK), Martin Tomášek (IS)	Added glossary
Mar 10, 2007	1.1	Martin Tomášek (IS)	Rework of chapter 9 (all sections).
Mar 17, 2007	1.2	Martin Tomášek (IS)	Added Executive Summary and short description of changes in the Introduction
		Dominik Smogór (EMA), Andrzej Marciniak (EMA)	Revision of chapter 9 (all sections)
Mar 21, 2007	1.3	Rolf Schillinger (UR)	Added annex A
		Marek Skokan (TUK), Ján Hreňo (TUK)	Rework of chapters 7 (all sections), 8 (all sections except 8.2.2) and 10 (all sections except 10.2.5 and 10.2.6)
		Karol Furdík (IS), Marek Paralič (IS)	Rework of chapters 5 (all sections), 8.2.2 and 10.2.6
Mar 25, 2007	1.4	Stefan Dürbeck (UR), Rolf Schillinger (UR)	Rework of chapters 6 (all sections) and 10.2.5
		Peter Bednár (TUK)	Added chapters A.1 and A.2
Mar 29, 2007	2.0	Tomáš Sabol (TUK), Marek Skokan (TUK), Peter Bednár (TUK)	Changes in chapter 5, 10.2.2 and annex A, revision of the text
		Karol Furdik (IS), Peter Bednár (TUK)	Changes in chapter 5.2.1.1 and 5.2.1.2

Files

Software Products	User files / URL
Microsoft WORD	

Content

Executive Summary

Functional specification provides description how a particular project or application should look like and work. It details what the finished product will do, how a user will interact with it, and what it will look like. We start with the description of the application in the report D3.1 Access-eGov Platform Architecture, where we described the overall functionality and software architecture. Our goal in this report is to provide a detailed description of the specific components of the system from the designer's perspective. We are using the following approach to prepare complete and usable functional specification:

- To define the application. In this stage we describe what the application is supposed to be, what the application is supposed to do and who is using this application. This stage was covered by the report D3.1.
- To create models. Various models will provide understanding of the components that make up the system. This stage is covered by the reports D3.2, D4.1 and D5.1.

We consider development of three different models:

- User conceptual model. Development of use cases and user roles is based on how the Access-eGov system is perceived by the user. This model of the system on the conceptual level is already described in D3.1.
- Designer model. This is where the interface components and relationships to be seen and experienced by the user requirements are defined. It details the available objects in the system and how it can use them to accomplish certain tasks. This model is presented in this report.
- Programmer model. Usually we create user and designer models, and then pass those off to the programmer who would then build the application. Every programming environment has inherent limitations and those constraints must be incorporated into the designer model. We propose description of specific implementation constraints related to selected implementation technology in the two subsequent reports D4.1 and D5.1.

The results contained in this report are modules of the Access-eGov platform and their components. Each module covers specific functionality of the system. The components of the module are specific steps (phases) how to fulfil the functionality of the module from the technical point of view. We develop use cases of each module considering its functionality, API of each component, and we describe the functionality of the module by sequences of interactions of the components. Further details of the components design considering implementation constraints of the selected technology are elaborated in the reports D4.1 and D5.1.

1 Introduction

1.1 Objectives and scope

The aim of this document is to provide a list of software modules and components that will be comprised in the Access-eGov architecture. A module is defined as a piece of software that is able to perform a specific functional task, working alone, independently of other modules. A component is defined as a piece of software that is able to perform a specific technical task. A component is not able to work alone, and to be useful it must be combined with other components to form software modules.

This document will describe in detail each of the modules on the list – providing for all of them a definition, function it performs within the overall architecture, components needed to implement the module, relations between those components, their communication, etc.

Following changes to the submitted version 1.0 for the first review were provided to this document:

- Additional details about functionality of modules and components were provided.
- Executive summary of the deliverable is included.
- Annex containing description of benchmarks, performance tests, scalability and simultaneous access of the used technology is included.

1.2 Document structure

- Definitions of used terminology
- A list of modules and high level definitions of the module functions
- Detail descriptions of identified modules together with their components

2 Definitions

Despite the fact that the terms like “component” and “module” are of common use the consortium (*at least during the phase of building common understanding inside of the consortium*) provides formal definitions of them.

2.1 Component definition

A software component is any piece of software that performs a specific technical task – i.e. it does not fulfil a user requirement by itself, but it solves a low-level problem specific to a domain. Although a software component performs a specific task, a component cannot operate individually without the tasks performed by other software components. This is due to the fact that a user requirement demands usually to solve a problem at different levels and in different domains and disciplines. Ideally a software component should only solve one specific problem so that it can be combined with other software components in many different scenarios. At the same time the software component API should be kept as simple and clear as possible, so that it can be integrated seamlessly with other components and a higher level reuse can be achieved.

To fully describe a software component, the following should be provided:

- Component name – a simple name to uniquely identify a software component. Besides, a short name will also be provided for practical reasons when writing and reading documents referring to that component.
- Component goal – the goal it pursues, i.e. the technical problem that the component solves. For each component its purpose inside the system or application must be clearly stated, it means, the added value that the component offers to the system developer.
- Component API – describes the input the component expects and the output it provides, what enables to view the component as a black box. Before starting coding any components its entire API must be closed and published, so that other components can use it. Once the component API has been published, software components can be grouped in modules where the components interact to fulfil one of the requirements of the system.

2.2 Module definition

A software module is any piece of software that performs a specific functional task – i.e. a task that fulfils a certain user requirement, thus solving a high-level problem specific to the user domain. A software module should be able to work independently of other modules and when the module stops working, other modules are not affected. Even when a software module can act individually, it is very common that modules performing different tasks are gathered together to build more complex software applications.

To fully describe a software module, the following information should be provided:

- Module name – a name uniquely identifying the software module. Besides, a short name will also be provided for practical reasons when writing and reading documents referring to that module.
- Module functionality – the functionality it adds to the software application, that is, all the system requirements it meets. Its purpose inside the whole system or application must be clearly stated, specifying what is the added value that module provides to the user.
- Software components – a list of software components that are needed to implement that module. Some of the software components may belong to one or more modules, but obviously there cannot be components that do not belong to at least one module.
- Software components sequence diagram – describes the way software components are gathered together and interact. The sequence diagram is a graph that represents interaction between different software components. At the highest level, a scenario from the use case describes a dialogue between the actor and the system. To support the dialogue, the system must have components that can take responsibility for each interaction. Mapping the sequential interactions to component operations is the job of the sequence diagram. Therefore to be able to provide a sequence diagram, the API of the software components needs already to be defined. The use of the sequence diagrams is a part of the UML meta-model of object-oriented modelling.

3 Logical architecture

The overall Access-eGov system (also called the platform) may be sub-divided into the three major parts [1]:

- the Access-eGov Infrastructure,
- the Access-eGov Personal Assistant client,
- Access-eGov Annotation services (not an integral, but an affiliated part of the Access-eGov Infrastructure).

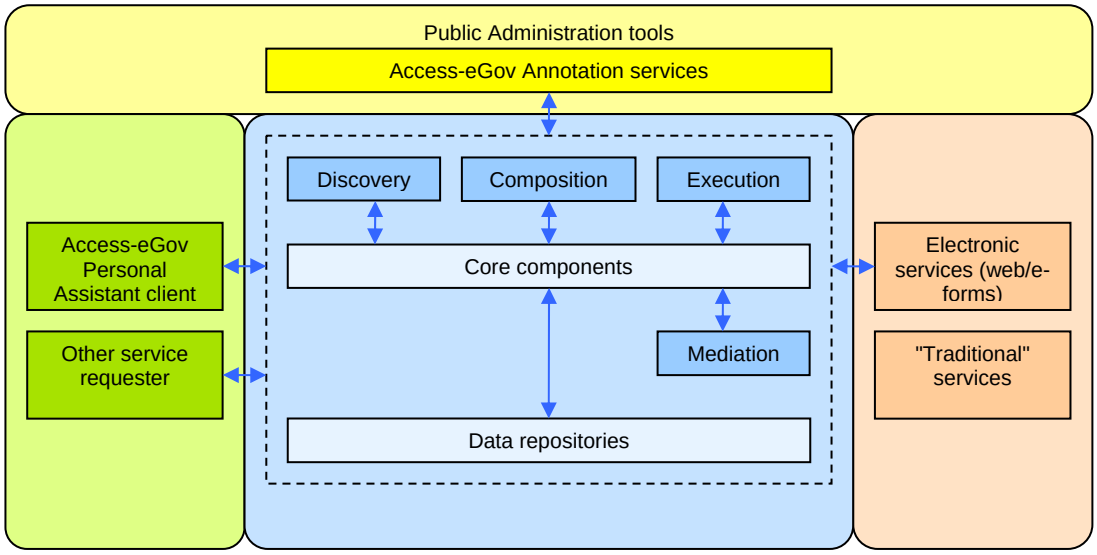

Figure 1 Overall architecture of the Access-eGov platform

The services are hosted and located on the premises of public administration institutions or their respective data centres. These services are simply made available via the Access-eGov system and thus they are not an integral part of the overall Access-eGov system. The services are provided either electronically (directly via web service interfaces or web forms) or in "traditional" (i.e. face-to-face) way – in this case they are only described and registered in the Access-eGov platform. Only executable services will dispose of an electronic XML-interface to the Access-eGov Infrastructure.

Public Agencies are supposed to annotate those services that they are willing to expose to the public. These kinds of service-related meta-data will be transferred to the Persistence layer via executable Core components. Therefore, domain experts may use a generic Annotation service component that will be available as a web-based application.

The Access-eGov Personal Assistant accesses the Access-eGov Infrastructure functionality via standardized interfaces and communicates with executable Core components that are charged with Discovery, Composition, and Execution of the registered public services. The Access-eGov Personal Assistant may only communicate with these Core components in order to gain access to the persistently held data.

Goal repository, Composed Model Repository, Service Repository, Domain Ontology Repository and Security Scheme Repository form components of the Persistence layer.

Public agencies may choose which of the above mentioned Access-eGov Infrastructure components they wish to install on their premises or data centres. Such a "local" installation of the Access-eGov Infrastructure components is supposed to interact as a peer in the peer-to-peer overlay network that Access-eGov is likely to consist of. The more components are installed locally (Core + Mediation + Persistence + Discovery + Orchestration + Execution), the more functional value a node will provide to the overall Access-eGov system.

Other service requesters (like SHG's Responsibility Finder) will also be granted access to the Access-eGov Infrastructure via XML-based interfaces.

4 Modules

This chapter will enumerate the modules that can be identified based on the reasoning and results described in user requirement analysis and the Access-eGov platform architecture. Most of the modules are derived from the needs implied by the use cases described in the deliverables D3.1 [1] and D2.2 [2], and are grouped in the following sets:

- Information provider related modules (proposed short name prefix MP-*)
- Information consumer related modules (MC-*)
- System management related modules (MM-*)

4.1 Information provider related modules

Information provider related modules cope with the following main tasks of information providers – namely annotating/registering services and building generic workflows out of already defined services.

The modules that belong to this category are:

1) Service annotation module
2) Service discovery module
3) Service composition module

4.2 Information consumer related modules

The modules related to information consumer deals with two main tasks. The first one is specification of a goal and execution of the retrieved services. The second one is the general access management of the Access-eGov platform services.

The modules that belong to this category are:

1) Scenario execution module
2) Personal assistant module

4.3 System management related modules

System management related modules cope with the core functionality of the Access-eGov platform.

The module that belongs to this category is: System core module.

5 Service annotation module

5.1 Module name and functionality

The proposed short name for the **S**ervice **Annot**ation Module is **MP-SAnnot**.

The Annotation service as used within the Access-eGov will consist of a web-based application that is not an integral part of the Access-eGov Infrastructure. Its main purpose is to enable domain experts to semantically describe their electronic/traditional services, by

using relevant public service ontology. This will explicitly involve annotating traditional websites as well. For this purpose, the web application provides spidering capabilities to allow easy inspection of the existing content, which concurrently can be annotated.

Via web service interfaces, the Access-eGov Annotation module is able to access the respective Repositories within the Persistence Layer (notably the Ontology and Service Repository) in order to register services and publish their descriptions. The creation, modification and editing of these semantic descriptions is controlled by the security subsystem.

The overall process of service annotation is depicted on the Use case diagram, presented on the Figure 2. On the side of public administration, a role of an Annotator is required. The Annotator should be a "domain expert", i.e. a person that is familiar with the domain(s) to be modelled, and also has some skills and experience in knowledge technologies.

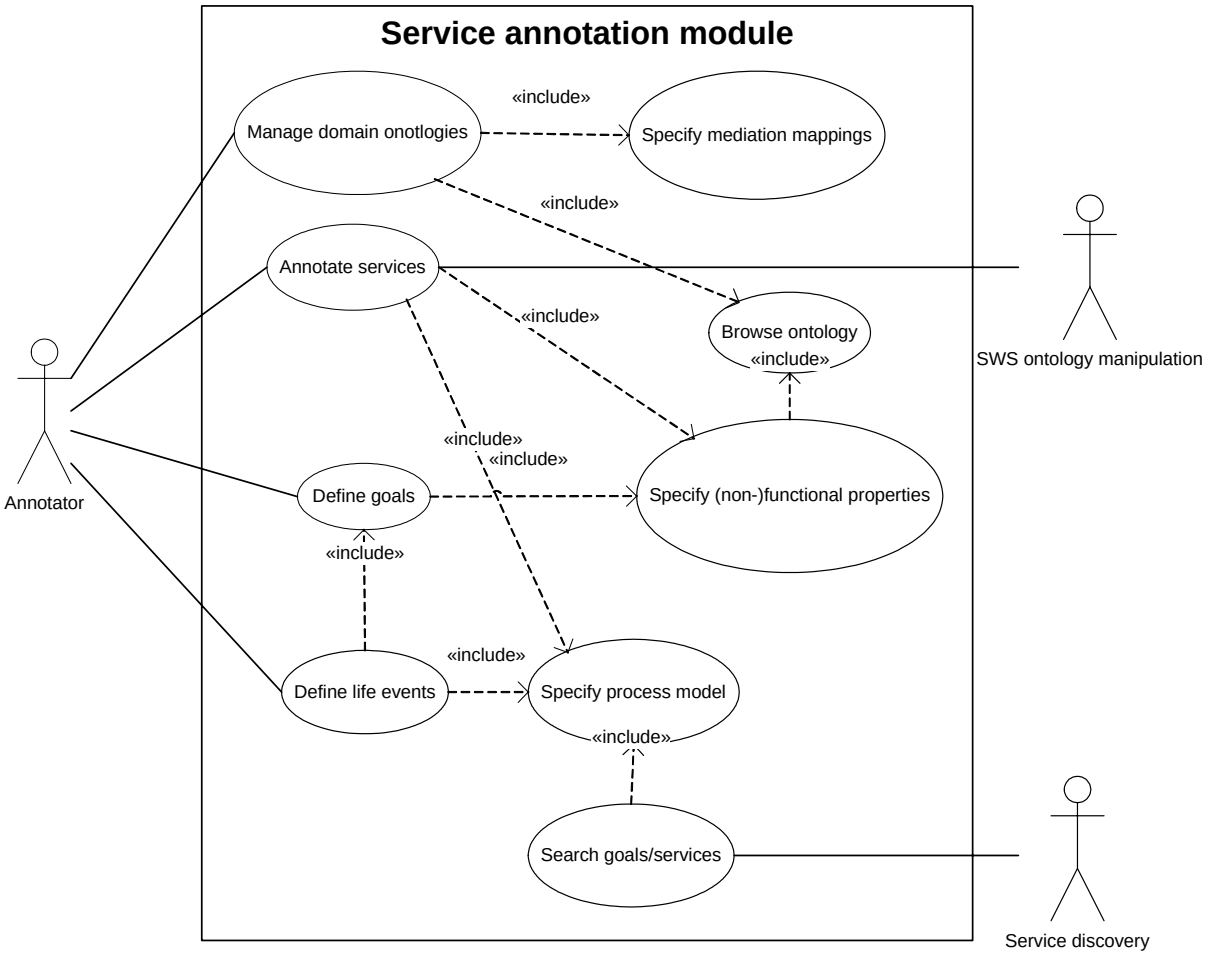

Figure 2 Use cases service annotation

Introduction of services (both electronic and traditional ones) to the AeG system requires the semantic description and consequent registration of the service. To semantically describe a service, the Annotator browses the space of available ontologies, stored in the AeG Ontology repository. Then, he/she uses the concepts and relationships from the selected ontology to mark-up important aspects of the service or website he/she is currently describing.

Availability of particular ontologies in a given time is usually not at the discretion of the Annotator, but governed by other, for example institutional, constraints.

In addition to the services, there is a possibility to create and manage the Goals and Life events in the MP-Sannot. Goals and Life events are workflow-like constructs that could be considered as outputs, or interfaces, provided by AeG system for users. The Annotator can define new or modify existing Goals by means of semantic description, similarly as it is done for services. Then, the Goals and services can be combined into the more complex workflow models – Life events, which are then exposed for users.

This is the first approach to describe the functionality related to annotation process. More details will be described in subsequent reports D4.1, D4.2 and D7.2.

5.2 Software components

The tasks and functionalities provided by the MP-Sannot (see Figure 2) can be divided into three separate groups. The first group is responsible for annotation of services, the second for definition and maintenance of Life events and Goals, and the third group covers the manipulation with domain ontologies.

According to this division, the following components were defined in the MP-Sannot module:

1) Service annotation component
2) Life events and Goals management component
3) Ontology management component

The MP-Sannot involves also some components from other AeG modules, namely:

From the MP-SDisc module (see chapter 6 for details):

1) Service Discovery component - used to find goals and services for the process model composition.

From the MM-SysCore module (see chapter 10 for details):

1) SWS ontology manipulation component – used to manipulate ontologies
2) WS connection manager component – used to connect infrastructure services
3) Security component – used to manage security
4) Notification services component – used to notify annotators

In the following subsections, the three inherent components of the MP-SAnnot module will be described in detail.

5.2.1 Service annotation component

5.2.1.1 Component name and goal

The proposed short name for the **S**ervice **A**nnotation Component is **SA**.

This component is used to create a service profile, which is a semantic description of particular service. The SA component uses the SWS ontology manipulation to browse the domain ontologies. The concepts and relations, taken from the domain ontology, then create the service profile – it means that they describe various properties of the service, its inputs and outputs and other characteristics (see report D7.2 for further details).

From the technical point of view, the wsmo4j library was taken as implementation platform for the SA component. Object representation of the service is provided by the *webService* class, defined in the wsmo4j. The service profile consists of functional (obligatory) and non-functional (optional) properties. The set of functional properties is given by SA implementation and can not be changed during the annotation process. Functional properties are specifications of input preconditions and output postconditions, and are expressed in the WSML logical expressions.

The non-functional (N-F) properties are semantic descriptions for the particular instance of a service. They describe semi-structured information intended for citizens for service discovery, e.g. service name, description, information about the service provider and properties which incorporate further requirements for service capability (e.g. traditional office hours and office location). Set of available N-F properties can be obtained from the domain ontology. When creating or modifying a service profile, the Annotator selects some of the N-F properties and specifies its instances by setting its value (as a WSML expression). In addition, the value of particular N-F property can be retrieved from an outer resource, e.g. from an existing web site. The requirement is that the web site will have unique URL and contains elements that can be uniquely identified. Then, the bound N-F property can retrieve the actual value of the web site element in run-time. This is a simple way how to semantically describe and involve a content of existing web sites into the AeG system.

5.2.1.2 Component API

Since the SA component interacts directly with annotator user and is proposed to be a web-based application, it does not expose any programming interface for other modules of the AeG system. Contrary, the SA component implements the interfaces of SWS ontology manipulation component, the WS connection manager component, and the Notification services component.

As it was stated above, the SA component is a web-based application dedicated for annotators to publish services. As such, it provides a web interface that is an integral part of the SA component. The simple schema of basic elements used in the web interface is depicted on the Figure 3.

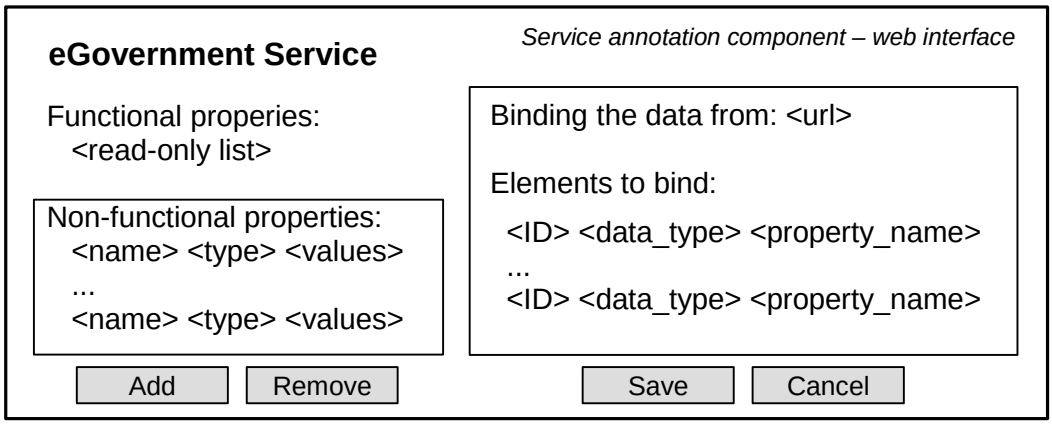

Figure 3: Schema of web interface for the SA component

The left side of the screen contains a read-only list of functional properties, and the editable list of N-F properties. The N-F properties are designed as name-type-value triples, where

proper values are given from domain ontology (expressed in WSML language), or can be entered manually.

The data for binding a particular N-F property with a source from an existing web site are listed in the right side of the screen. The elements for binding are uniquely identified within the web site by its ID. In addition, the elements are specified by proper data type and a name of the N-F property. Then, the bound N-F property can retrieve its value from the element of the outer web site. If the web site or particular element is not available, then the property will take the type and value from its default specification, listed in the left panel of the screen.

5.2.2 Life events and goals management component

5.2.2.1 Component name and goal

The proposed short name for the **L**ife **E**vents and **G**oals **M**anagement Component is **LEGM**.

The MP-SAnnot module will have a component for the management of life events and goals. This information will be stored in separate repositories and later on will be accessed by the Personal Assistant module to enable more user-friendly and effective navigation in life events and goals. This component will provide the functionality to:

1) Describe atomic (simple) goals. This will include specification of the functional properties required to achieve this goal and specification of the non-functional properties, which will additionally constrain candidate services.
2) Compose goals to the more complex generic scenarios to specify life events. User interface for the design of the generic scenarios will be based on the graphical notation and will allow intuitive visual design of the workflow and dataflow of the process model. The same interface can be reused to specify orchestration process model for the composed services.

The process of describing atomic goals by functional and non-functional properties is similar as in the case of SA component, however, a different data repository is used for the LEGM. The SWS ontology manipulation component is used to browse the Life events ontology and to select proper concepts and relations to constitute the description of an atomic goal (see report D4.1 for further details).

Object representation of an atomic goal is provided by the *goal* class, defined in the wsmo4j. The functional (obligatory) and non-functional (optional) properties, also defined in the wsmo4j library, are used to express the content of goals.

Atomic goals can be composed into more complex structures – generic scenarios that can then be instantiated as life events. The generic scenario is a workflow-like sequence of goals and sub-goals; it specifies a process model that consequently guides a decomposition of the life event to atomic elements.

5.2.2.2 Component API

Again, the LEGM component interacts directly with annotator user, so as such, it does not expose any programming interface for other modules of the AeG system. As a consumer, the LEGM implements the interfaces of SWS ontology manipulation component, the WS connection manager component, the Notification services component, and the Service composition module.

The user interface for description of atomic goals and for composition of goals into the generic scenarios is proposed to be a stand-alone Java application. The design of service-goal

sequences will be based on the graphical notation to allow intuitive visual manipulation of the workflow and dataflow of the process model.

5.2.3 Ontology management component

5.2.3.1 Component name and goal

The proposed short name for the **Onto**logy **M**anagement Component is **ONTOM**.

The Ontology Management Component is responsible for the management of the Access-eGov domain ontologies used to represent the functional and non-functional properties of a particular service. This functionality includes:

1) Build ontologies from scratch or by reusing existing ontologies imported from the Ontology repository.
2) Specify mediation mappings to other ontologies.

The following ontology types are assumed to be used within the AeG system:

Ontology name	Domain
Service Ontology	Atomic processes, their combination and constraints
Administration Ontology	Organisational structure and contact information; Responsibilities
Legal Ontology	References to laws and legal requirements
Document Ontology	Content, structure and inputs of forms, documents
User Case Ontology	Needs and goals of citizens / businesses, including individual constraints
Life Event Ontology	Description of life events / business episodes, including constraints

In the process of creation of these ontologies, the existing ontology resources will be analysed in detail and used as much as it will be possible. The examination of available ontology resources and its usability will be done within the T7.3 and T.1.

Management of all domain ontologies will be handled by a WSMO-compliant ontology editor. Natural choice is the WSMO Studio, which provides the ontology editor with rich functionality, including support for ontology format conversion, choreography designer, mediation mappings, WSML validator, etc.

5.2.3.2 Component API

The ONTOM component again interacts directly with annotator user, so it does not expose any programming interface for other modules of the AeG system. As a consumer, the ONTOM implements the interfaces of the SWS ontology manipulation component, the WS connection manager component, the Notification services component, and the Security module.

User interface is given by the WSMO Studio application that can be installed as a stand-alone application, or as a plug-in to the Eclipse SW package.

6 Service discovery module

6.1 Module name and functionality

The proposed short name for the **S**ervice **Disc**overy Module is **MP-SDisc**.

In the process of service discovery, functional properties of goals and services are semantically matched by the Access-eGov Discovery module to select services, which are able to achieve these goals. Non-functional properties specified by the requester are then used to additionally filter or reorder the discovered services according to the requester preferences.

Service discovery in Access-eGov can be sub-divided into the two cases “Full-text search” and “Semantic search” for service descriptions. The diagram below shows detailed citizen use cases serviced by different components of MC-PAssist with involvement of other Access-eGov components.

The description of actors follows below:

- **Personal Assistant:** The Personal Assistant takes user input and feeds it to the Full-Text Search engine.
- **Full-text search** functionality will be provided in form of an interface to already existing full-text search engines in order to retrieve services and life events/goals from simply comparing the set of properties they’re annotated with.
- **Full-text matching** will be used to compute the low-level matching against properties and includes pattern matching functionality
- **Get Goals**: The user asks the Personal Assistant to retrieve one or more goals that will be matching a given term
- **Filtering** is used in all cases to order the result set according to a given scheme (relevance, alphabetic order, etc.)
- **Mediation**: since Access-eGov will broker services across organisational boundaries with (in some cases) a multitude of different ontologies, mediation will be needed to map terms of source ontology to target ontology.
- **Reasoning** can be used after Mediation to judge about conformity of different input and output types. It can also be invoked as step for filtering.

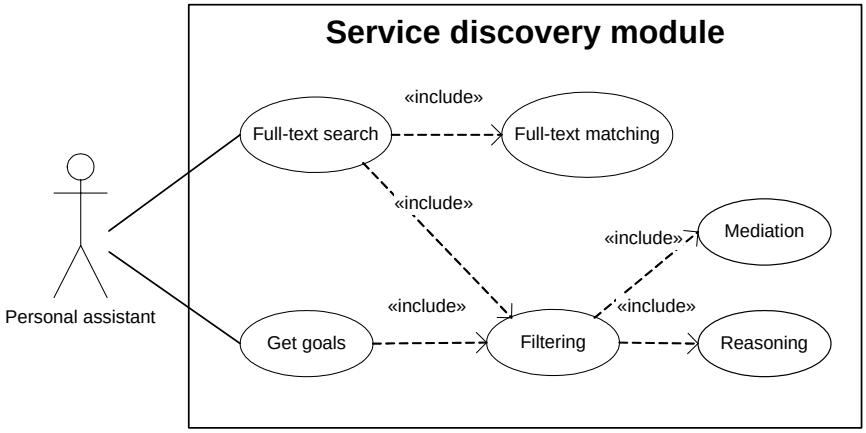

Figure 4 Use cases full-text search for service discovery

- **Semantic Search**: In the case of semantically valuable user input (a specific input-output set is requested), Access-eGov can go beyond mere full-text matching for a semantic on-the-fly computation of an appropriate chain of services. The user input is then used to check preconditions and effects of registered services without invoking an already existing workflow.
- **Ontology Manipulation** is required when repositories will need to be looked up for registered entries.

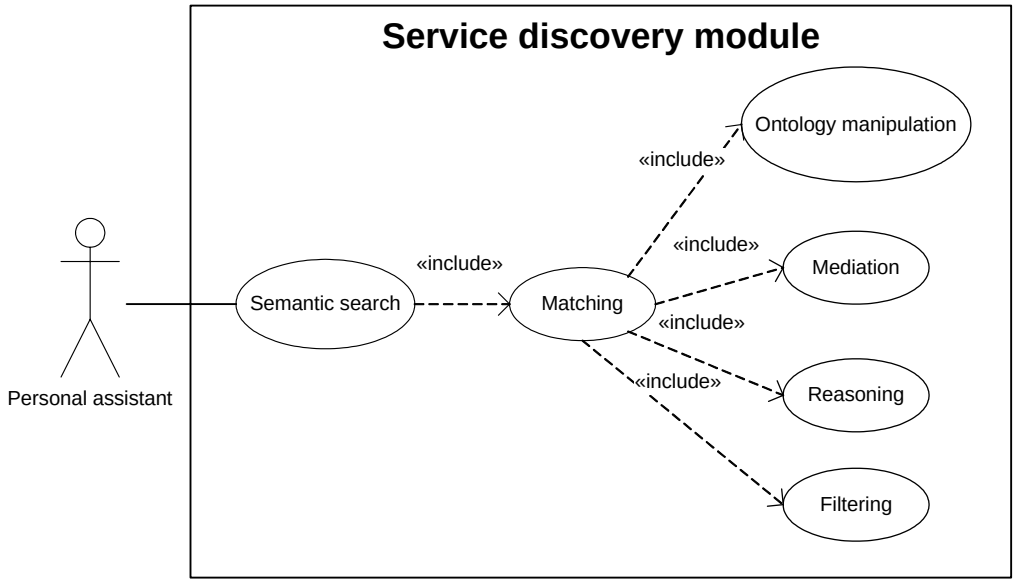

Figure 5 Use cases semantic search for service discovery

6.2 Software components

The components defined in this module are:

1) Full-text search component
2) Matching component
3) Filtering component
4) Reasoning component

5) Mediation component

Components involved in this module are defined in the module MM-SysCore (see chapter 10 for details) and may include:

1) SWS ontology manipulation component – used for looking up data repositories
2) WS connection manager component – used to connect to Infrastructure Services
3) Security component – used to manage access control to repositories and services

6.2.1 Full-text search component

6.2.1.1 Component name and goal

The proposed short name for the **F**ull-**T**ext **S**earch Component is **FTS**.

Access-eGov's high-level full-text search component will provide an interface to the full-text search of services and life events/goals. Full-text queries will be matched to the text extracted from unstructured non-functional properties (i.e. name, description, etc.). Full-text index of the various entities will be managed in the corresponding data repository. Additionally the component enables obtaining list of goals that are relevant for requesting user. It is provided a form suitable for processing by non knowledge-enabled clients.

6.2.1.2 Component API

Function	Description
Full-text search goal/service (query, non-functional properties): Service/Goal	Provide full-text search for goals and services. This functionality takes a query string and looks up repositories connected to the Access-eGov network for goal and service descriptions matching the query string or the non-functional properties that the triggering process fed in to FTS. **Inputs:** query - full text query; non-functional properties - used for filtering and reordering of results **Outputs:** Set of matching services or goals
Get goals(non-functional properties): Set of goals	Provides a filtered list of all goals. The triggering process (or Personal Assistant) invokes the functionality by feeding non-functional properties in. Given this input, FTS lets ONTOM (cf. chapter 10) look up data repositories and FIL filter the returned result set. **Inputs:** non-functional properties - used for filtering and reordering of results **Outputs:** Set of all goals, filtered by supplied properties

6.2.2 Matching component

6.2.2.1 Component name and goal

The proposed short name for the **Mat**ching Component is **MAT**.

Depending on the available information and the complexity of the semantic annotations of the services, semantic matching of the services can be based on the following strategies:

- *Simple semantics*: matching is based only on the outputs and effects, which are specified as logical expressions with terms defined in the domain ontologies. Terms of the requested goal are semantically matched to the terms of the outputs and effects provided by the services with possible inference for exploring the knowledge about the output types. This approach does not capture the actual relation between service input and the corresponding outputs. Thus, the semantics of a requested service is only described in a conceptual manner.
- *Rich semantics*: in case that the actual input values are available for the discovery process, matching can be based on rich semantics, which captures the relation between the service inputs and outputs. The input data provided in the discovery request by the user or from the other services are used to check preconditions and effects without the invocation of the existing services. This way, Access-eGov Discovery can achieve highest precision, because it is possible to select only services, which are expressingly able to provide the requested outputs with the actual inputs.

Which strategy will be used depends on the availability of the input data. For example, rich semantic discovery should take place at execution time, when the Access-eGov Execution module can collect as many data as it will be possible for the Access-eGov discovery.

The Matching component will directly make use of the SWS ontology manipulation component in order to look up data repositories, namely the Ontology and Services Repository.

6.2.2.2 Component API

Function	Description
Search service (goal, non-functional properties): Service	Provide semantic search of services based on the matching of functional properties specified for the requested goal. Given a selected goal and/or non-functional properties that a service description shall contain, MAT is triggering a semantic search for finding appropriate services. **Inputs:** goal - requested goal; non-functional properties - used for filtering and reordering of results **Outputs:** Set of matching services, filtered using FIL

6.2.3 Filtering component

6.2.3.1 Component name and goal

The proposed short name for the **Fil**tering Component is **FIL**.

When the Matching component select services according to the functional properties, non-functional properties specified by the requester are then used to additionally filter or reorder discovered services according to the requester preferences. The decision about the type of processing – i.e. filtering/reordering is taken based on the request context specified in the discovery request.

6.2.3.2 Component API

Function	Description
Filter (services, non-functional properties): Services Filter (goals, non-functional properties): Goals	Filter collection of services or goals according to the non-functional properties. Different schematic orders can be applied for filtering, e.g. sorting by relevance, alphabetic order, etc. **Inputs:** services/goals - filtered collection of services/goals; non-functional properties - used for filtering and reordering of results **Outputs:** Filtered set of services/goals

6.2.4 Reasoning component

6.2.4.1 Component name and goal

The proposed short name for the **Reas**oning Component is **REAS**.

To explore the domain knowledge about the input and output types, Discovery components will invoke a reasoner component. Reasoning will be used also for filtering the services according to the structured non-functional properties. Since efficient reasoning for some functional and non-functional properties will require optimized procedures (for example, to infer nearest geographical location), the interface for reasoning should allow plug-in extensions based on various implementations. More details will be specified in the subsequent project reports (D4.1 and D4.2).

6.2.4.2 Component API

Component API will be specified in the subsequent project reports (D4.1 and D4.2).

6.2.5 Mediation component

6.2.5.1 Component name and goal

The proposed short name for the **Med**iation Component is **MED**.

The role of the Access-eGov Mediation component is to reconcile the semantic and data heterogeneity that can appear during discovery, composition or execution. For the Access-eGov platform, it is not expected that the public administrations will use the same common ontology to describe their services or life events related to these services. It is possible that each organization can have its own domain ontology, which has to be mapped to other ontologies used to describe goals and services. Mappings of the domain ontologies can be required during all phases of the request, namely during the discovery, composition and execution phase.

The mediation will be based on mappings expressed with the mapping ontology and designed for the mapped domain ontologies. Mappings can be stored in the ontology repository together with the corresponding mapped ontologies. Access-eGov Mediation will load and translate mappings into the rules, which will be used to merge ontologies and to translate specified incoming instances from the input ontology to the instances of the target ontology.

The following types of mediators were identified:

- *Goal to life event mediation* - In case that one life event refers to many goals, each goal can be described with different ontology. All involved ontologies will be merged to form a union ontology which will be used to specify the generic scenario process model associated with the life event.
- *Service to goal mediation* - When the Access-eGov Discovery component matches semantic description of the service with the semantic description of the goal, service-to-goal mediator will align different goal and service ontology. This mediation can be integrated in the reasoning interface, which will dynamically merge goal and service ontology in the matching phase for both functional and non-functional properties.
- *Life event to service mediation* - This is the reverse mapping from the union ontology, which specifies the orchestrated scenario for the life event request to the ontology used to describe particular service involved in the orchestration. This mediation is required for the invocation because grounding mechanism is specified for the service ontology only. In this case, the involved mediator will transform data instances from the source scenario ontology to the target service ontology for the inputs and in the reverse direction for the outputs.

6.2.5.2 Component API

Component API will be specified in the subsequent project reports (D4.1 and D4.2).

6.3 Software components sequence diagram

The sequence diagrams in this section illustrate the interactions between MP-SDisc and MP-SComp. These interactions identify the functions necessary for MP-SDisc. For each of the cases, a sequence diagram is provided.

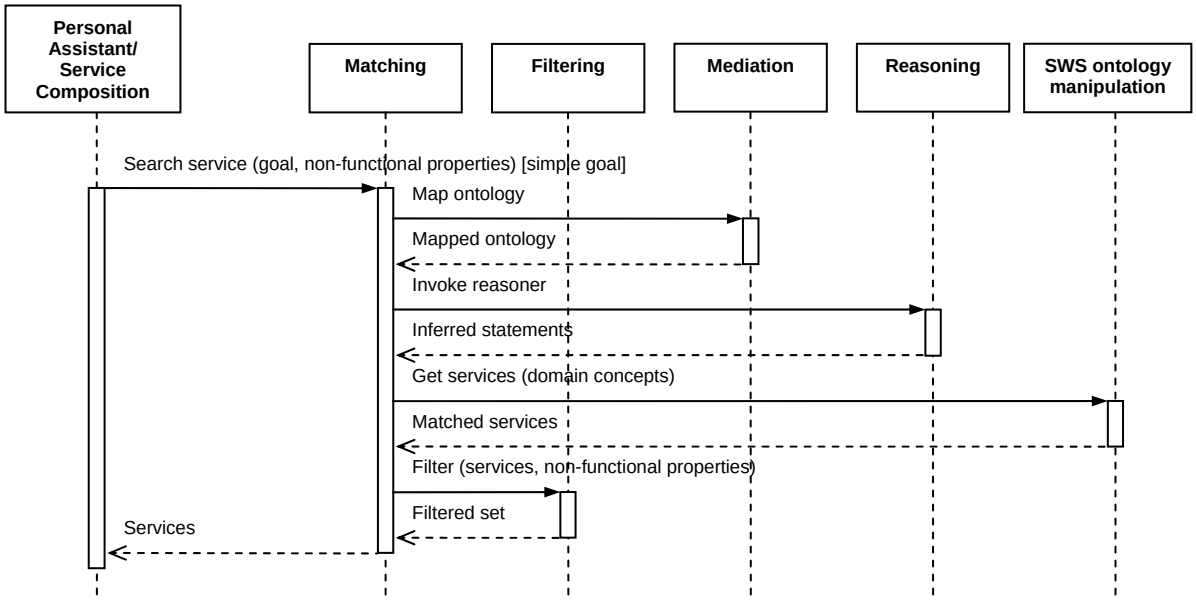

Figure 6 Sequence diagram for goal discovery given a goal and non-functional properties

The sequence diagram on Figure 6 describes in detail the case of a service search according to a goal and some non-functional properties:

1. **Search Service (goal, non-functional properties) [simple goal]:** The Personal Assistant sends a Search request for a service given a simple goal to MAT.
2. **Map Ontology**: MED is used to mediate between different ontologies that are used to describe entities (i.e. services or goals, etc.). Sufficient ontology information is transferred to MED.
3. **Mapped Ontology**: An appropriate (in the best case 1:1) mapping scheme for two ontologies will be returned to MAT.
4. **Invoke reasoner**: REAS is invoked in order to judge about appropriate equivalence of properties, inference of terms, etc. based on the input and the service request in question.
5. **Inferred statements**: REAS returns its findings and assumptions about equivalence to MAT.
6. **Get services (domain concepts)**: based on these findings, ONTOM is asked for retrieval of matching service and goal descriptions in question.
7. **Matched services**: ONTOM returns the matching storage entries to MAT.
8. **Filter (services, non-functional properties)**: FIL is invoked for sorting the randomly bundled result set.
9. **Filtered set**: FIL returns the set of matching services filtered according to a given schematic order (relevance, alphabetic order, etc.).
10. **Services:** A Set of Services matching the request is returned to the Personal Assistant

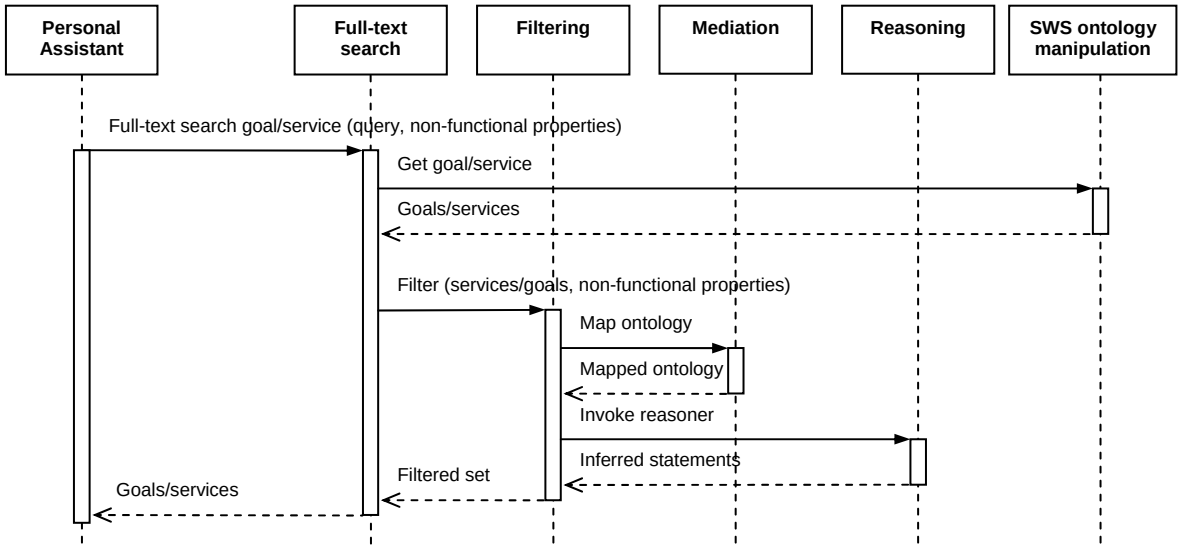

Figure 7 Sequence diagram for service/goal discovery given a query string and non-functional properties

The sequence diagram on Figure 7 describes in detail the case of simple full-text search (simple retrieval by a query string):

1. **Full-text search goal/service (query, non-functional properties):** The Personal Assistant sends a Search request for a service or goal to FTS.
2. **Get Goal/service**: FTS sends a full-text search request to ONTOM.
3. **Goals/Services** matching the query string are returned in random order to FTS
4. **Filter (services/goals, non-functional properties)**: FIL is invoked for sorting the randomly bundled result set.
5. **Map ontology**: in order to sort the result set, MED is called to semantically assist in mapping between the different source ontologies
6. **Mapped ontology**: the mapping is returned to FIL for further sorting computation.
7. **Invoke reasoner**: REAS is invoked in order to judge about appropriate equivalence of properties, as a last step in filtering.
8. **Inferred statements**: REAS returns its findings and assumptions about equivalence to FIL.
9. **Filtered set**: FIL returns the set of matching services and/or goals as filtered according to a given schematic order (relevance, alphabetic order, etc.).
10. **Goals/Services:** A Set of Services matching the query string is returned to the Personal Assistant

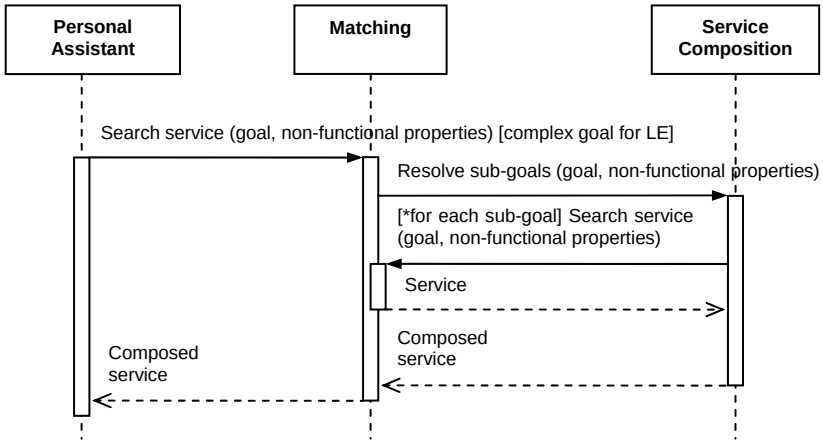

Figure 8 Sequence diagram for service discovery given a goal and non-functional properties.

The sequence diagram on Figure 8 describes in detail the case of service resolving and discovery during a complex goal scenario (retrieval by a complex goal):

1. **Search service (goal, non-functional properties) [complex goal for LE]:** The Personal Assistant sends a Search request for the services that are affiliated to a complex goal to MAT.
2. **Resolve Sub-goals (goal, non-functional properties)**: MAT sends a full-text search request to MP-SComp.
3. **Search Service (goal, non-functional properties)**: for each possible sub-goal, the found service descriptions are returned to MAT.

4. **Service:** The services of the various sub-goals are ordered and composed by MP-SComp to form a reasonable cascade.
5. **Composed Service**: The service composition/workflow is returned to MAT and to the Personal Assistant respectively.

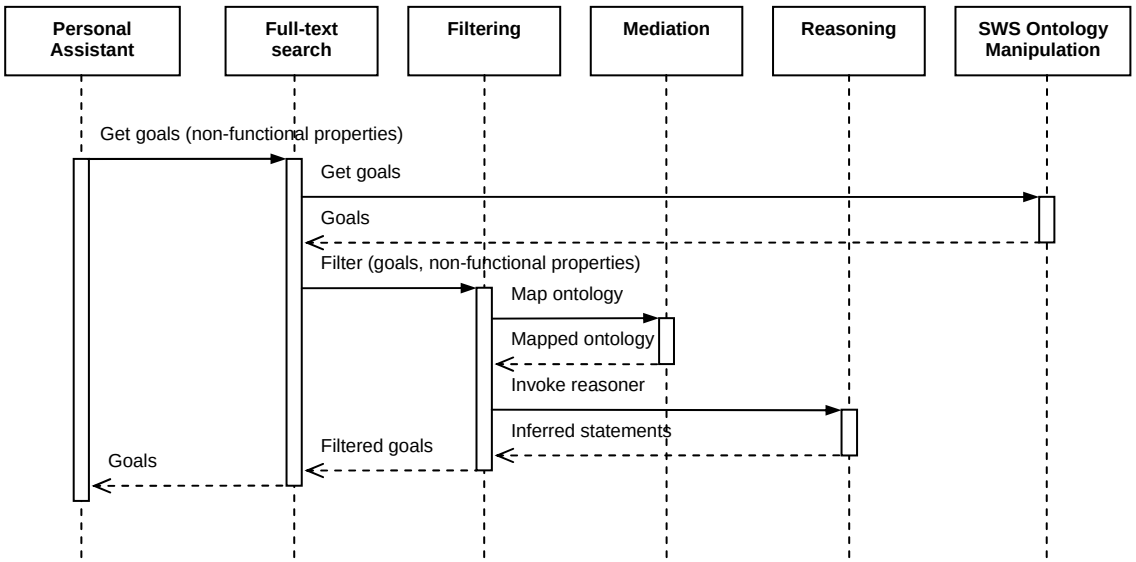

Figure 9 Sequence diagram for goal discovery given non-functional properties

The sequence diagram on Figure 9 describes in detail the case of a simple full-text search of goals:

1. **Get Goals (non-functional properties):** The Personal Assistant sends a Search request for a goal to FTS that shall match the selected non-functional properties.
2. **Get Goals**: FTS sends a full-text search request to ONTOM.
3. **Goals** matching the request are returned in random order to FTS
4. **Filter (goals, non-functional properties)**: FIL is invoked for sorting the randomly bundled result set.
5. **Map ontology**: in order to sort the result set, MED is called to semantically assist in mapping between the different source ontologies
6. **Mapped ontology**: the mapping is returned to FIL for further sorting computation.
7. **Invoke reasoner**: REAS is invoked in order to judge about appropriate equivalence of properties, as a last step in filtering.
8. **Inferred statements**: REAS returns its findings and assumptions about equivalence to FIL.
9. **Filtered goals**: FIL returns the set of matching goals as filtered set according to a given schematic order (relevance, alphabetic order, etc.).
10. **Goals:** A Set of Services matching the non-functional properties in question is returned to the Personal Assistant

7 Service composition module

7.1 Module name and functionality

The proposed short name for the **S**ervice **Comp**osition Module is **MP-SComp**.

In case that Access-eGov Discovery module cannot find an atomic service, which is able to achieve requested goal, goal description is delegated to the Access-eGov Composition module, which will try to orchestrate existing services to the new scenario to solve this goal. Although there are many initiatives to define industry standard languages for web service orchestration like BPEL, they have restricted capability to support only static service composition. Access-eGov Composition module provides support for dynamic composition of the services, which is not based on the static workflow pre-defined for the life event.

For the dynamic service composition, the following three classes of the problems were identified:

- *Fulfilling preconditions*: a service that can provide the desired effects and outputs exists, however, not all of this service's preconditions or inputs are met from the outset.
- *Generating multiple effects*: requester encodes in the goal multiple effects that are related, yet can be generated by different services.
- *Dealing with missing knowledge*: some information required to select services is missing at the composition time.

Automatic composition of the services, which will solve these problems, is the subject of the current and future research. For this reason, current specification of the Access-eGov Composition component includes a semi-automatic approach based on the generic scenarios defined for the life event categories.

Generic scenario will specify the process model, which will guide decomposition of the multiple effects specified for the life event to the sub-goals achieved by the atomic services. Sub-goals are then resolved with the Access-eGov Discovery, which will match sub-goal description with the existing services, possibly with the additional planning in the Access-eGov Composition to fulfil preconditions. Discovery matching will be based on the functional properties specified for the sub-goal and non-functional properties specified either by the user in the request, or constrained by the logical expressions predefined in the generic scenario. With the resolution of the sub-goals, generic scenario can be dynamically customized according to the specific user needs and conditions.

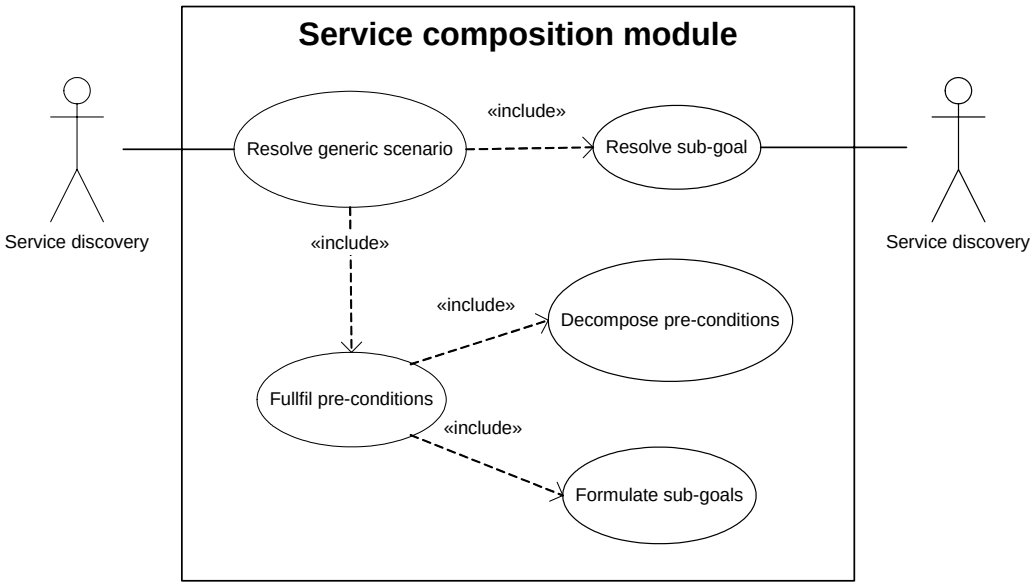

Figure 10 Use cases service composition

The description of actors follows:

- **Service discovery.** Service discovery is responsible for discovery of services according to their properties. It is a general user of MP-SComp representing one component from MP-SDisc. This MP-SDisc's component delegates unresolved goal description to the MP-SCom and also it receives back the sub-goals from MP-SComp.

The description of the use cases follows:

- **Resolve generic scenario.** After receiving the description of unresolved goal it is necessary to check whether the orchestration interface is specified. If yes, the identified sub-goals are sent back to MP-SDisc for resolving. If no, the Fulfil pre-conditions case is included.
- **Fulfil precondition.** When unresolved goal doesn't contain orchestration interface the MP-Comp tries to fulfil its precondition by chaining of the existing services (by recursively regarding service's effects and outputs).
- **Decompose pre-conditions.** To find services which provide necessary preconditions specified by goal, these preconditions are decomposed to subsets.
- **Formulate sub-goals.** By resolving goal without orchestration interface the process model in form of sub-goals with specified precondition is built and therefore a new orchestration interface is defined.
- **Resolve sub-goal.** When a goal is decomposed to the sub-goals these sub-goals are sent back to the MP-SDisc where they are resolved by services.

7.2 Software components

The components defined in this module are:

1) Resolving component
2) Chaining component

The component involved in this module defined in the module MP-SDisc (see chapter 6 for details):

1) Mediation component

The components involved in this module defined in the module MM-SysCore (see chapter 10 for details):

1) SWS ontology manipulation component - used to manipulate ontologies
2) WS connection manager component - used to connect infrastructure services
3) Security component - used to manage security

7.2.1 Resolving component

7.2.1.1 Component name and goal

The proposed short name for the **Res**olving Component is **RES**.

The RES component is responsible for replacing sub-goals by the relevant services, i.e. it is responsible for service composition. The RES component might also delegate goal to the Access-eGov Chaining component.

The delegation of goal is done in case some preconditions of a goal are not fulfilled. We can let the decision whether to use the service of Chaining component in such situation or not to the user, but we will assume that it will be automatically delegated to the Chaining component.

Sub-goals might be specified in the orchestration interface (a process model of the generic scenario as abstract activities) and they have to be resolved before the scenario is executed. To resolve sub-goals, the RES component invokes the Access-eGov Discovery module. Note, that in case the goal was delegated to the Chaining component (process model is not explicitly specified and the relevant service can not discovered) the Access-eGov Discovery module has to be used (see the section about Chining).

7.2.1.2 Component API

There is only one API function for the Resolving component. This API will resolve all sub-goals formalized as a complex goal with the internal process model which consists of abstract activities. A complex goal has defined sub-goal and its activities in its choreography interface. As can be seen in the following table, the inputs consists of goal - formalizing generic scenario specified for the life event - and non-functional properties used for filtering of candidates (services) for sub-goals. The provided output is in a form of composed activities which fulfil sub-goals in the generic scenarios or a service consisting of more atomic services in a not-necessary linear sequence - composed service.

Function	Description
Resolve sub-goals (goal, non-functional properties): composed service	Resolve generic scenario to the composed service. This API will resolve all sub-goals of the specified generic scenario (formalized as a complex goal with the internal process model which consists of abstract activities) and create new composed service. It will be used in case the complex goal has to be resolved. To

	resolve the complex goal means to find relevant atomic services fulfilling the sub-goals and compose them into composed service. **Inputs:** goal – complex goal delegated from the discovery module during the (semantic) service discovery process non-functional properties - used for filtering of candidates for sub-goals **Outputs:** composed service form of composed activities which fulfil sub-goals in the generic scenario

7.2.2 Chaining component

7.2.2.1 Component name and goal

The proposed short name for the **Chain**ing Component is **CHAIN**.

The responsibility of the CHAIN component is to compose services. The strategy is to check whether the effects and outputs of one service meet the preconditions and inputs of the following one. It aims at fulfilling all the preconditions which have to be met.

In case that not all the inputs or preconditions of the resolved services are met from the outset, "chaining" of the services can be used to overcome this problem. Chaining can be understood as a way to compose services by recursively regarding the effects and outputs of one service as the preconditions and inputs of a following one until a desired effect is reached. This can be done automatically with the AI planning methods. Decomposition with the generic scenarios should further simplify this process.

7.2.2.2 Component API

There is only one API function for the Chaining component. As can be seen in the following table, it takes the service which can not provide its outputs because some of its preconditions are not resolved. The provided output is a new process model specified in the orchestration interface in a new goal and composed services - composed atomic services which fulfil sub-goals in the new goal.

Function	Description
Resolve preconditions (service): process model	Compose services by recursively regarding the effects and outputs of one service as the preconditions and inputs of a following one until the preconditions (a logical expression in the functional properties) of the specified service are not met. The Resolving component will be accessing this function of the Chaining component in case that not all the inputs or preconditions of the resolved services are met. **Inputs:** service which preconditions have to be resolved (a service can not provide its outputs because of these unresolved preconditions)

	Outputs: process model (the orchestration interface in a new goal) with composed services (composed atomic services which fulfil sub-goals in the new goal)

7.3 Software components sequence diagrams

Following sequence diagram illustrates the interactions between MP-SDisc and both component of MP-SCom. These interactions identify necessary functions of MP-SCom's components.

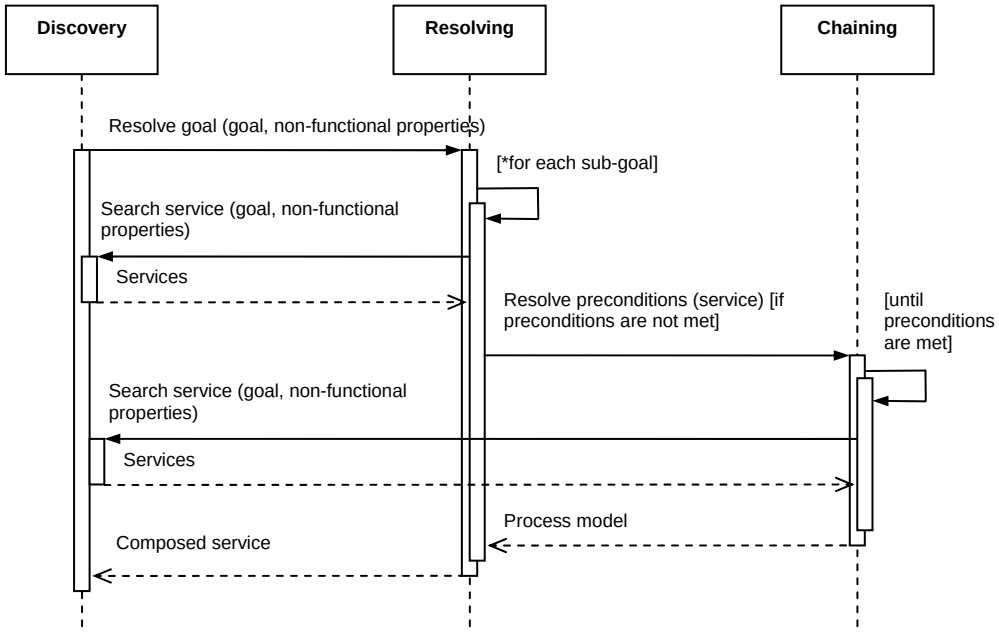

Figure 11 Sequence diagram for service composition

Description of sequences follows:

1) **Resolve goal.** MP-SDisc sends an unresolved goal to Resolving component (RES). If the goal has an orchestration interface each sub-goal defined in this interface is sent back to the MP-SDisc.
2) **For each sub-goal**. Each sub-goal from complex goal is processed in the following.
3) **Search service.** Sub-goals are sent with its non-functional properties to the MP-SDisc where they are replaced by relevant services. In case there is no service, which match the sub-goal MP-SDisc send it to RES.
4) **Services.** Discovered services are returned to RES.
5) **Resolve preconditions.** In case the service relevant to the particular goal was not found the RES delegates it to the Chaining component (CHAIN), where outputs and effects of the existing services (MP-Disc is again called here see **Search service**) are recursively regarded (the chain of services is built) until all preconditions of specified goal are met.

6) **Until preconditions are met.** The CHAIN first decomposes the set of precondition and then tries to discover relevant service for each of this sub-set of preconditions.
7) **Search service.** Similar as 3
8) **Services.** Similar as 4
9) **Process model.** If the goal is resolved, the process model in the form of new orchestration interface (this model can be reused in the future).
10) **Composed service.** Finally the composed service is built. From the services available to resolve goals.

8 Scenario execution module

8.1 Module name and functionality

The proposed short name for the **Sc**enario **Exec**ution Module is **MC-ScExec**.

When the user wants to achieve his goal, he lets the personal assistant start the execution of the retrieved service or workflow. Progress of this run is always visible to the user through the personal assistant client. So, simply, the user executes the scenario via Personal Assistant. Execution interaction can be the execution itself, secondly inspecting the state of the execution and finally updating the missing information during the execution process. As we can see in the Use case of this component on the following figure there are 3 above mentioned main activities of this component.

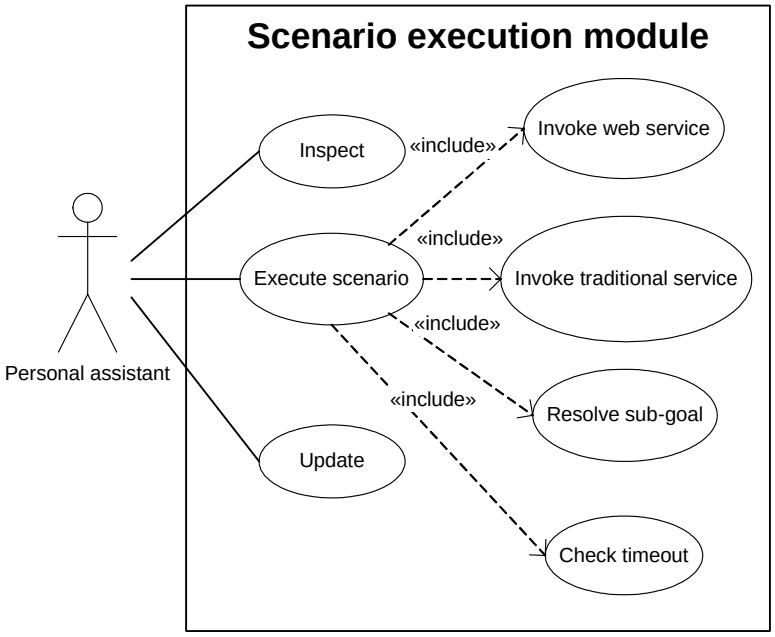

Figure 12 Use cases scenario execution

Main activity is the "Execute scenario", which includes the following activities:

- Invoke web service
- Invoke traditional service

- Resolve sub-goal
- Check timeout

Invoking the web service is the process in which the context is processed into the inputs of the particular web service which is then invoked. Outputs of the web service are then processed and returned to the system.

Invoking of the traditional services is the special type of invocation, where activity is led back to the user (to the Personal Assistant). Executing of the scenario waits then for user to input the output of the traditional service.

Resolving of sub-goals is needed if not all the scenario for the execution is fully resolved at the beginning. It means some of the steps in the scenario are already the services to execute, but some of the steps are still only goals (it is possible these goals can be resolved after some of the services in the scenario are executed). Thus additional resolving is needed during the scenario execution.

If the service invoked has specified a delivery time, a timeout-timer for output-delivery will be generated and can be checked. If the service does not provide output during the necessary time, timer raises the time-out event.

In many cases, the execution phase of such services or scenarios will also include activities that are only available offline. Access-eGov will, in this case, simply wait until notification (update of process context) of completion of the specific offline activity. That's why Inspect and Update activities are needed in the use cases diagram.

8.2 Software components

The components defined in this module are:

1) Goal/scenario execution component
2) WS invocation component

The components involved in this module defined in the module MP-SDisc (see chapter 6 for details):

1) Mediation component

The components involved in this module defined in the module MM-SysCore (see chapter 10 for details):

1) SWS ontology manipulation component - used to manipulate ontologies
2) WS connection manager component - used to connect infrastructure services
3) P2P connection manager component - used to connect P2P infrastructure - distributed repositories
4) Security component - used to manage security
5) Notification services component - used to send timeout notification to Personal Assistant client

8.2.1 Goal/scenario execution component

8.2.1.1 Component name and goal

The proposed short name for the **G**oal/**S**cenario **E**xecution Component is **GSE**.

The Goal/scenario execution component is responsible for invoking composed services of orchestrated scenarios. For simple web services Goal/Scenario Execution Component simply uses the WS Invocation component (see next subchapter) to execute it. For composed services, the Execution component executes process instances of the specified process model and executes it with initial input data provided by the service requester (D3.1). The inputs to this invocation are initial process context and composed services. As the web services can be executed asynchronously (there is timer for them) and some of the services in the scenario can be of the traditional type (user by himself modifies the input context, so the execution has to wait for user to update the context) there are the update and inspect functions to provide needed functionality.

8.2.1.2 Component API

Function	Description
Execute service/scenario (service/scenario, initial context): Process context	Personal Assistant client uses this API to execute scenarios (composed services) or to invoke atomic services. If the input is atomic web service, it is simply invoked via WS Invocation component. If the input is the scenario, which is the set of the composed services, this function tries to invoke the services one after other (as they are composed and outputs of one can be inputs of other). **Inputs:** service/scenario; initial process context which specify inputs from the user profile **Outputs:** Process context
Update (context update): Process context	Personal Assistant client uses update API in case that invoking of traditional service has changed the associated process context. If the scenario execution consists of the traditional services, this function is used to update the context so the web services in currently executed scenario, which are waiting for the outputs of the traditional services (which is the input for them) can be invoked. It is also notified by the timeout event if the execution of some service timed out. **Inputs:** Process context - with updated data **Outputs:** Process context
Inspect scenario: Process context	Personal Assistant client uses inspect API for inspecting current state of the scenario execution. It is used to handle asynchronous character of the scenario execution. It inspects the state of the timers for the services and the state of their input and output data. **Inputs:** Process identification **Outputs:** Process context
Create timer (Timer)	If the service has the time in which it is needed to be executed, the timer is created for it.

	Inputs: service

8.2.2 WS Invocation component

8.2.2.1 Component name and goal

The proposed short name for the **W**eb **S**ervice **Inv**ocation Component is **WSInvC**.

The invocation component is responsible for invoking atomic web services in the proper manner. In case the elementary web service has to be invoked, this component performs it. The inputs to this invocation are process context and web service identifier. Invocation component processes context information into inputs of web service (grounding). It also processes the output data of the web service into output data needed by Goal/scenario execution component.

The tasks to be done within WS Invocation component follow from the fact that the execution process remembers the state and data of the execution itself in the form of process and ontology instances. These have to be transformed into right series of messages with the appropriate web service. In order to invoke a web service not only the flow of the messages should be specified, but also the data (input, output) contained in messages should be transformed. Transformation is done according predefined rules that connects ontology instances (used in execution process and defined in WSML) and data in messages (used by web service and defined in WSDL).

8.2.2.2 Component API

There is only one API function for the Web Service Invocation component. As seen in following table. It takes as an input the identifier of the web service and the input data. It first grounds the input data, transforms the input data from the internal representation of our system to the form required by the web service. After the input data are grounded, the web service is invoked via web protocol with these data. Invocation of the web service returns (again via web protocol) the output data. These data are then grounded again from the web service related form to the form needed by our system. Output data in the form needed by our system are returned as a result of the Invoke web service function.

Function	**Description**
Invoke web service (service, input data): Output data	Goal/scenario execution component uses this API to invoke Web service. Supposing the needed grounding information is available (defined by the service annotator), the invocation component communicates with the web service to be invoked (service) with appropriate message flow. Data used in messages are transformed from ontological instances (input data) into XML data and from XML data into ontological instances (output data) - when the web service sends back a message. Output data are put into the execution process context that defines the next actions. **Inputs:** service **–** semantically annotated service to be invoked; input data – process execution context data relevant for the web service invocation, that are in the

	form of ontology instances **Outputs:** Output data – data to be updated in the process context execution (in the form of ontology instances) as result of the web service invocation (created after the grounding process from XML data that receives the invocation component from the Web service)

8.3 Software components sequence diagrams

Following sequence diagram illustrates the interactions between Personal Assistant and all components of the Scenario execution module. These interactions identify necessary functions of these components.

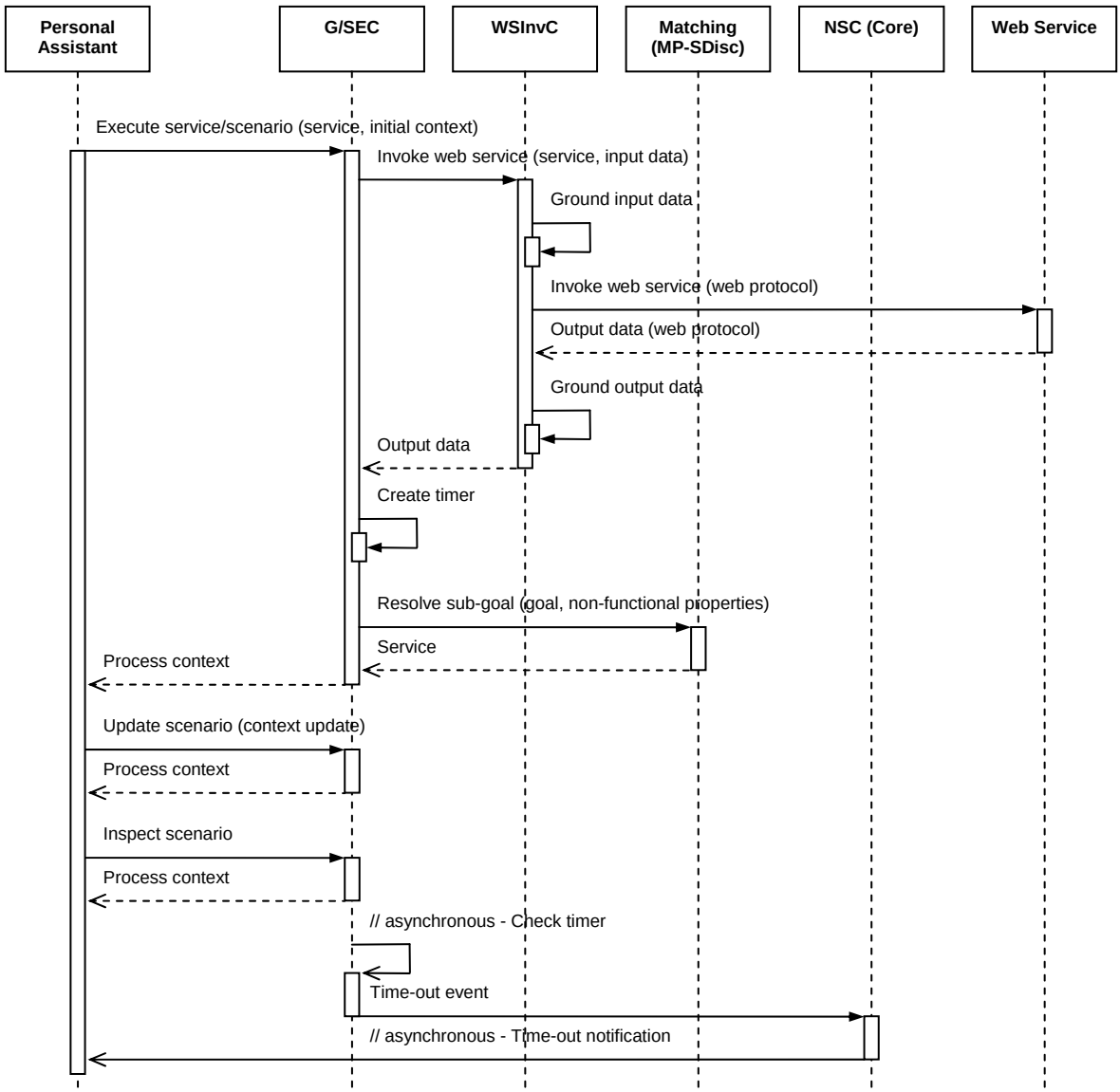

Figure 13 Sequence diagram for scenario execution

Description of sequences follows:

1) **Execute service/scenario (service, initial context).** Personal assistant initialises the execution of the scenario/service with the initial context. It sends the initial context and scenario or service to the Goal/Scenario Execution Component.
2) **Invoke web service (service, input data).** Goal/Scenario Execution Component invokes the Web Service Invocation Component with the service and input data.
3) **Ground input data.** Web Service Invocation Component grounds input data (processes context information into inputs of web service)
4) **Invoke web service (web protocol).** Via web protocol the Web Service Invocation Component invokes web service(s)
5) **Output data (web protocol).** Output of the web service is sent back to the Web Service Invocation Component.
6) **Ground output data.** In the Web Service Invocation Component output of the web service is grounded (web protocol based output is processed to output data for the system).
7) **Output data.** Web Service Invocation Component sends back the output data to the Goal/Scenario Execution Component.
8) **Create timer.** If the service invoked in the process activity has specified a delivery time, the Goal/Scenario Execution Component will create a timeout-timer for output-delivery.
9) **Resolve sub-goal (goal, non-functional properties).** If some of the goals are still not resolved, after some execution time of the scenario execution they can be resolved into services in the Service Discovery Module.
10) **Service.** Service Discovery Module returns resolved service to Goal/Scenario Execution Component.
11) **Process context** of the resolved sub-goal is returned to the Personal Assistant.
12) **Update scenario (context update).** If the context has to be updated from the Personal Assistant (in case that invoking of traditional service has changed the associated process context), context update is sent from the Personal Assistant to the Goal/Scenario Execution Component.
13) **Process context.** is returned to the Personal Assistant after the scenario was updated.
14) **Inspect scenario**. If Personal Assistant wants to inspect the scenario execution, it calls the Goal/Scenario Execution Component to inspect scenario.
15) **Process context** is returned to the Personal Assistant of the current process execution state.
16) **// asynchronous - Check timer.** Goal/Scenario Execution Component checks in the asynchronous mode the timer set for the execution.
17) **Time-out event.** It is generated, if the executed service runs out of time without returning the result.
18) **// asynchronous - Time-out notification.** Goal/Scenario Execution Component notifies the **N**otification **S**ervice Component in the Core module about time out of the service execution.

9 Personal assistant module

9.1 Module name and functionality

The proposed short name for the **P**ersonal **Assist**ant Module is **MC-PAssist**.

The personal assistant module is responsible for interaction with the user of the Access-eGov platform. The user (usually citizen) does not work directly with the MC_PAssist module, but connects through Personal Assistant Client that is a web application. The Personal Assistant Client communicates to the Access-eGov platform and invokes its services through the MC-PAssist module. Citizen must be at first authenticated through his/her user name and password. Authentication is important for the Access-eGov application, because the supported e-government services are customized according citizen's profile which may contain sensitive data. The citizen connected to the platform can use following general functions of the MC-PAssist module:

- Manage citizen's user profile. Citizen's user profile is stored in the data repository and is partly editable by the citizen. Citizen's user profile is also used to fill in non-functional properties when discovering services and customizing a goal for particular user. More details on the user profile and data repository can be found in D5.1.
- Select goals/scenarios. The main purpose of the Access-eGov is the support composing e-government services into a specific life event scenario. The data repository of the Access-eGov platform stores simple goals scenario based complex goals. Citizen can either browse the list of all goal/services or select one by a full-text search.
- Execute scenario. Citizen orders selected goal to be executed. He/she enters required data and the execution process starts. The process of an execution and its results must be visualised and presented to the citizen. The visualisation must consider that the execution process is discrete and each step can take a log time and citizen is disconnected during that time.

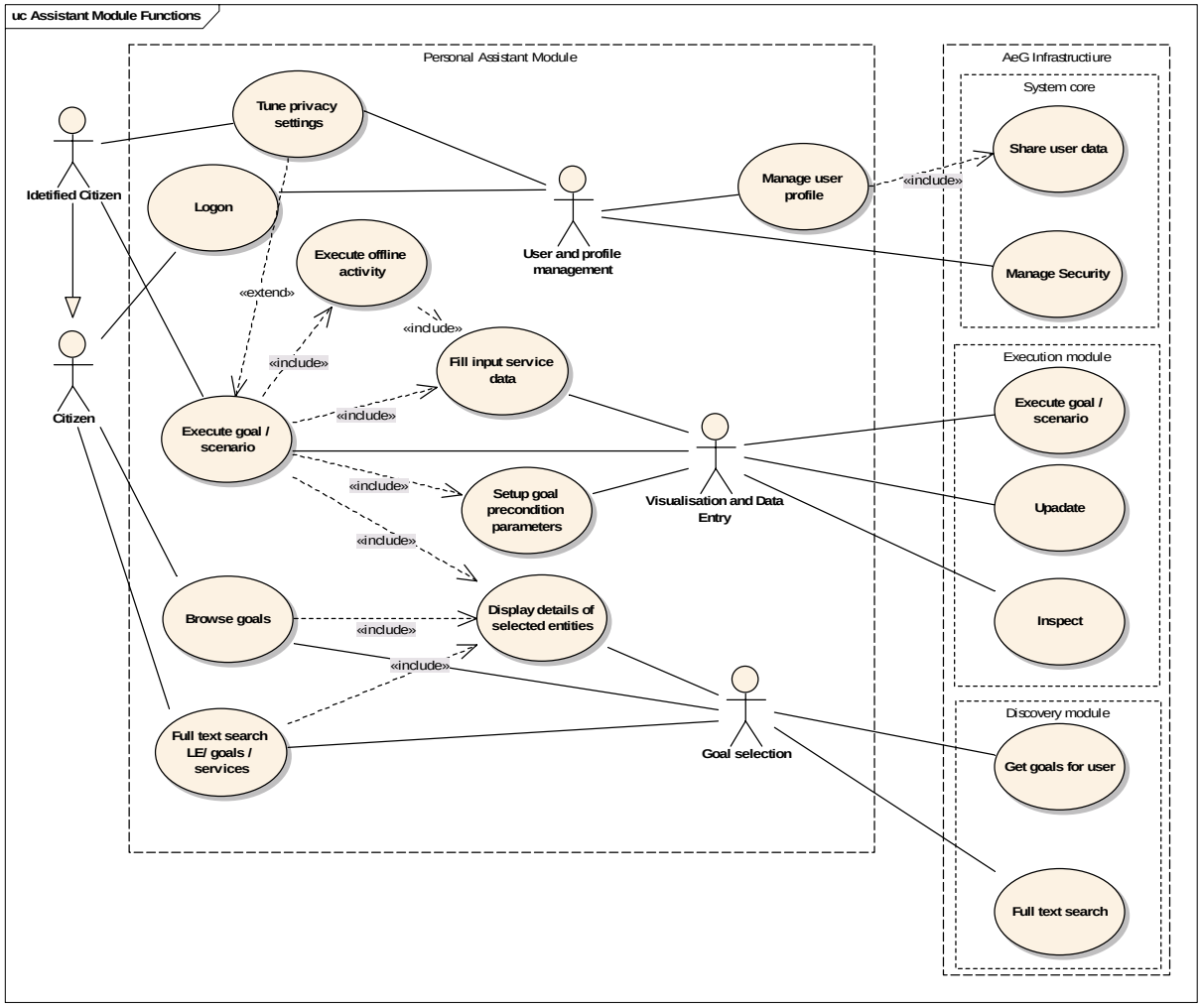

Figure 14 Use cases personal assistant

More detailed specification of the Personal assistant module will be presented in subsequent reports D5.1 and D5.2.

Diagram shows detailed citizen use cases serviced by different components of MC-PAssist with involvement of other Access-eGov components.

The description of actors follows:

- **Citizen.** A general user which connects to the system and must be authenticated first. The only function provided for that user is to search for services and watch services details, create a user profile. No other functions can be used by that user.
- **Identified citizen.** This citizen is already registered and he/she can logon to the system. After logon he/she can manage his/her user profile, tune its privacy settings, search or browse goals and execute a scenario associated with the goal. When the scenario is running the citizen can always see the status of its execution and whole execution process is visualised.
- **User and profile management.** This actor represents a component of the MC-PAssist module. The purpose of this component is to provide an interface (UI and server component) for managing information associated with the citizen. The details of the component are described in the next section.

- **Visualisation and data entry.** This actor represents a component of the MC-PAssist module. The purpose of this component is to provide visualisation and enable user to interact with the processed scenario. The details of the component are described in the next section.
- **Goal selection.** This actor represents one component of the MC-PAssist module. The purpose of this component is to provide selection of the goal or scenario either by browsing or full-text search. The details of the component are described in the next section.

The description of use cases follows:

- **Tune privacy settings.** The citizen is provided with a set of forms. The forms are filled in by the citizen with his/her personal data. This data is then used to define user profile of the citizen and is stored in the data repository for the customization of goal execution process and for further management.
- **Logon.** When a citizen has registered its user profile in the Access-eGov system it can logon. After logon, the citizen is authenticated and he/she is able to use Access-eGov functions and execute scenarios which are customized according to his/her user profile.
- **Execute goal/scenario.** After selection of desired life event/scenario the citizen can execute discovered e-government services required to fulfil the scenario. Execution of the services requires citizen input data and data from the citizen's user profile. State of execution of each activity is displayed for the citizen.
- **Browse goals.** The citizen is provided with a catalogue of goals/life events from the repository. The catalogue is prepared according ontology of goals/life events. The catalogue includes all possible goals that the citizen can execute within the Access-eGov platform.
- **Full-text search.** The browsing of goals is only one possibility how to select desired goal. Another possibility is to search life events repository. The module supports full-text search, which means the citizen enters keywords that are used in discovery module for the goals retrieval.
- **Execute offline activity.** Some of the services are traditional and offline activities from the point of view of the Access e-Gov platform. The citizen is provided with location of the desired office and prepares input forms/applications for the officers. When the citizen finishes the offline activity it can again connect and continue the execution process by entering the results of the service.
- **Fill input service data.** When citizen executes the desired scenario each service within the executed services workflow needs some data as an input. The data can be used from citizen's user profile, but there is still possibility that the service needs additional information. Citizen is provided by the set of input forms to enter desired data.
- **Display details.** Citizen can always see the details of each goal or service on the screen. The details include general information and guide concerning a goal/service: office procedure description, contact information, required documents, payment conditions, etc.
- **Setup goal preconditions.** Before the details of a goal to be executed can be watched and its workflow examined the citizen needs to customize it filling a form. The form contains information which is necessary to juxtapose workflow fitting user's case and initiate the goal execution.

- **Manage user profile.** User profile data is entered by the citizen. Module provides input forms for the citizen. Input forms are processed and entered data is stored in the data repository. Part of this data can then be updated by the citizen.

9.2 Software components

The MC-PAssist module is divided into three different components. Each component is responsible for the specific types of functionalities of the module to fulfil user requirements.

The components defined in this module are:

1) User and Profile Management component
2) Goal Selection component
3) Visualization and Data Entry component

The components involved in this module and defined in the module MP-SDisc (see chapter 6 for details):

1) Full-text search component – used for full text search
2) Get goals – used to get the list of goals for browsing

The components involved in this module and defined in the module MC-ScExec (see chapter 8 for details):

1) Goal/scenario execution component – manages scenario execution which is visualized by the assistant

The components involved in this module and defined in the module MM-SysCore (see chapter 10 for details):

1) SWS ontology manipulation component – used to get information necessary for creating structure of goals during goal navigation
2) Data Repositories – used as actual data store for user related data
3) WS connection manager component – used to provide communication channel between the assistant and core services
4) Security component – used to manage security of users who log in and execute services
5) Notification services component – the assistant passes notifications requested by the notifications component to the citizens.

9.2.1 User and profile management component

9.2.1.1 Component name and goal

The proposed short name for the **U**ser and **P**rofile **M**anagement Component is **UPM**.

This component performs tasks related to the user profile management. It is responsible for the management and access to user related data called user profile. User profile contains personal data of the user and data gathered from previous service executions. For the user personal data the user ontology is used. It is stored in central data repository. The component communicates also with Repository management component which is responsible for data repositories manipulation.

The UPM keeps history information from previous service executions too. List of reusable resources maintains a set of concepts from the domain ontology, especially filled forms and documents resulting from already processed services. Data repositories are used for actual storage.

Citizen connected to the Access-eGov is authenticated and authorized to access his/her profile within the UPM component. It is to keep secure access to Personal Assistant Client for citizens who decide to create user profile and keep their sensitive data on Access-eGov platform. The authentication method will meet requirement set by law for keeping and providing such data by information systems.

Moreover the component provides forms that allow users to provide different types of credentials that are processed by the security module and passed on to government services.

9.2.1.2 Component API

The component interacts directly with the user and does not expose any API usable by the rest of the platform.

9.2.2 Goal selection component

9.2.2.1 Component name and goal

The proposed short name for the **G**oal **S**election Component is **GS**.

This component is responsible providing user interface for navigation and selection of life events, goals and services. Two main possibilities of goal selections are supported for the user. The first is full-text search of goal/life events or services. The second one is browsing of goals from the organized structure (catalogue). The component should also enable to display details of those entities in a user comprehensible manner.

- Full-text searching allows identification of desired life events or goals based on matching contents of non functional properties containing human language descriptions to a list of keywords given by the citizen. Every result in the result set shows a short excerpt from the matched text.

Properties of services and goals that can be matched are as follows:

- Explicit keywords, a title and primary description of goals/life events. Life events and goals are mixed in the result list.
- Selected functional properties of a goal (like outputs) can be also matched e.g. documents produced matched by title or descriptions. In such case, an indication which property leads to its matching is present.

On the other hand the goal browsing enables to select a single goal/life event from a hierarchical catalogue. One non-functional property associates a goal or any of its outputs to one of taxonomy classes that are defined in goal and domain ontologies. The hierarchical structure of those classes forms a base for the catalogue.

9.2.2.2 Component API

The component interacts directly with the user and does not expose any API usable by the rest of the platform.

9.2.3 Visualisation and Data Entry component

9.2.3.1 Component name and goal

The proposed short name for the **V**isualisatio**n** Component is **VN**.

This component implements functionality related to visualisation and interaction during the whole process of scenario execution with citizens. In particular the below listed are types of tasks that are considered.

Displaying information related to selected goals/scenarios. The component should enable to display details of selected entities in a user comprehensible manner. The details include general information about goal/service, procedure description, contact information, required documents, payment conditions, etc.

Visualising workflow structure of LE scenarios/composite services. The citizen is able to see overall process of the scenario. The list of services must be shown with corresponding input and output documents which are required for the execution process.

Visualising state of (atomic/composite) service execution. The citizen must be able to see the current state of the execution. Many of the services takes long time and the disconnection of the citizen from the Access-eGov system must be allowed during the execution in any state. Selecting particular services if many of them are used to fulfil the selected goal. There can be more services discovered to fulfil the goal. The citizen must be able to select from the list of services one that will be executed afterwards.

Entering data that is matched to goal preconditions before forming a workflow structure that can be browsed by the user. The citizen is provided by a input form to enter goal preconditions if they are needed.

Constructing and allowing the citizen to fill in data forms that are then feed to executed services. Similar interaction takes place during traditional (offline) service execution.

9.2.3.2 Component API

The component interacts directly with user and does not expose any API usable by the rest of the system.

9.3 Software components sequence diagrams

Following sequence diagram shows the interactions of the citizen and the components of the MC-PAssist module in the Access-eGov platform. The MC-PAssist components do not have any API usable by the other components of the platform, but their interactions with the rest of the system are defined here. The interactions clearly identify the functionality of each component from the MC-PAssist module.

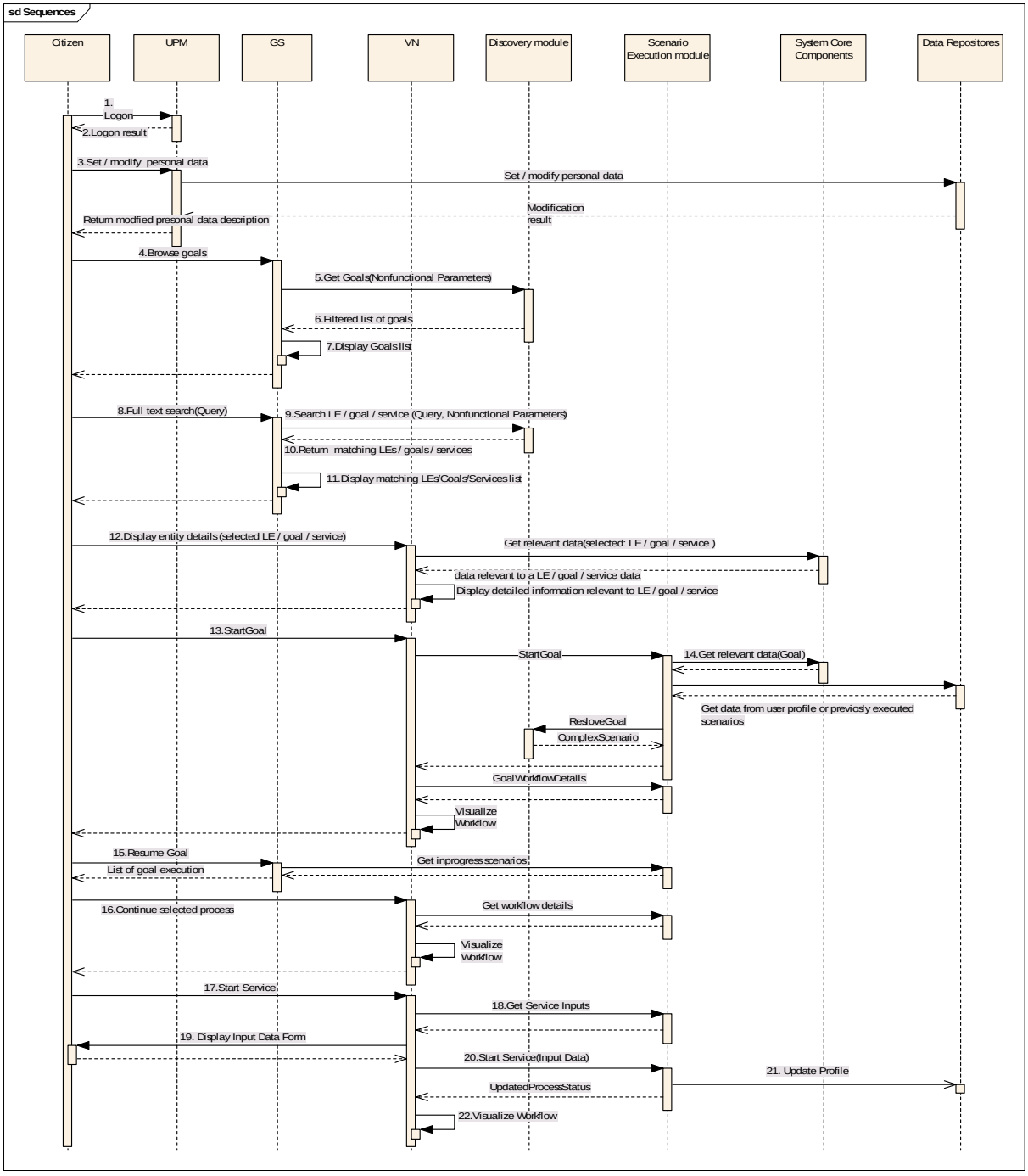

Figure 15 Sequence diagram for personal assistant

Description of sequences follows:

1) **Logon.** Citizen logs on to the system through the UPM component. Citizen passes its authentication data (username and password) to the UPM component.
2) **Logon result.** The UPM component informs citizen about successful or not successful authentication of the citizen. Unsuccessful logon could lead either to create a new user profile or to publish some general security information. Successful logon allows citizen to use the Access-eGov functions.

3) **Set/Modify personal data.** The citizen can either create new or update own user profile. The UPM component provides forms for filling personal data. The data are passed to DR component which is responsible for storing it in the data repository. The return from the UR component gives the information about successful or not successful access to the data repository. The UPM component informs the citizen about storing the personal data and gives general privacy policy information.
4) **Browse goals.** Citizen is requesting the catalogue of goals from the GS component. The UPM component prepares the hierarchical list of goals that can be viewed and browsed in a user suitable manner.
5) **Get goals.** The GS component interacts with components from the Service discovery module. The components in the module are responsible to for retrieving all life events and structuring them to the hierarchical catalogue according taxonomy of the domain ontologies.
6) **Goals.** The components from the Service discovery module return the hierarchical list of goals to the GS component. The GS component processes the list for publishing to the citizen.
7) **Display goals.** The GS component prepares the view screen from the hierarchical list of the goals discovered by the components of the Service discover module. The view screen represents the catalogue of the goals that a citizen can use for browsing.
8) **Full-text search.** The citizen sends a query to search for goals (or services) to the GS component. The query is a simple keyword that could be matched by some properties assigned within the goals (or services) stored in the ontologies.
9) **Full-text search goal/service.** The GS component interacts with the components from the Service discovery module. The components in the module are responsible for matching the search query (and non-functional properties) and preparing the result list of goals (or services).
10) **Return Goals/services.** The components from the Service discovery module return the list of matched goals (or services) to the GS component.
11) **Display matched goals/services list.** The GS component prepares the view screen from the result list of the goals (or services) discovered by the components of the Service discovery module through matching entered keywords. The view screen shows the result list with the details of matching properties and the user can select one of them for execution or examination.
12) **Display entity.** The User wants to see details regarding selected goal or service. VN component retrieves data related to selected entity from DR and prepares an information page compiled from non-functional properties and displays it to the user.
13) **StartGoal.** When the citizen orders to execute displayed goal a new instance of process is created at the execution module. The citizen expects the VN component display browsable workflow of services for the selected complex goal. The VN retrieves data needed for visualization from the execution module.
14) GSE component gets complex Goal definition and data from user profile so it can build the process context. Then it invokes service discovery module in order to obtain a composite service.
15) **Resume goal.** Execution of a goal usually takes long time and user can be disconnected from the system during the execution plan. After returning to the system user is able to

resume the executed workflow to continue next steps of the action plan. First, the user has to choose which of already process instances he/she would like to continue.

16) **Continue process.** The VN component is asked by the user to resume selected process. As in case of new execution GSE component provides workflow details which are then visualized, so that the user can continue interaction with the process.

17) **Start service.** While in execution of Goal user may decide to execute a particular service.

18) **Get inputs.** VN asks GSE for list of input information pieces needed for a service. Part of them can be inferred from the process context.

19) **Fill form.** Basing on the missing information report from GSE, VN generates a input form which is displayed to the user to fill. Fields inferred from process context are already filled.

20) **Execute service.** Input information is passed on to GSE which executes the service.

21) **Update profile.** New information about the user, extracted from the user profile updates the user profile. Then the information about process update is returned to VN.

22) **Visualise scenarios.** The VN component provides view screens for the citizen to display details about execution status of the whole scenario and each its single step. The component keeps track of the execution during the whole process and is able to display historical or current data to the citizen in any time the citizen is connected to the system.

10 System core module

10.1 Module name and functionality

The proposed short name for the **Sys**tem **Core** Module is **MM-SysCore**.

The system core module is responsible for the interaction with the user of the Access-eGov platform and also for tasks, which are relevant to the core platform functionality. That means manipulation with ontologies, connections management and security issues.

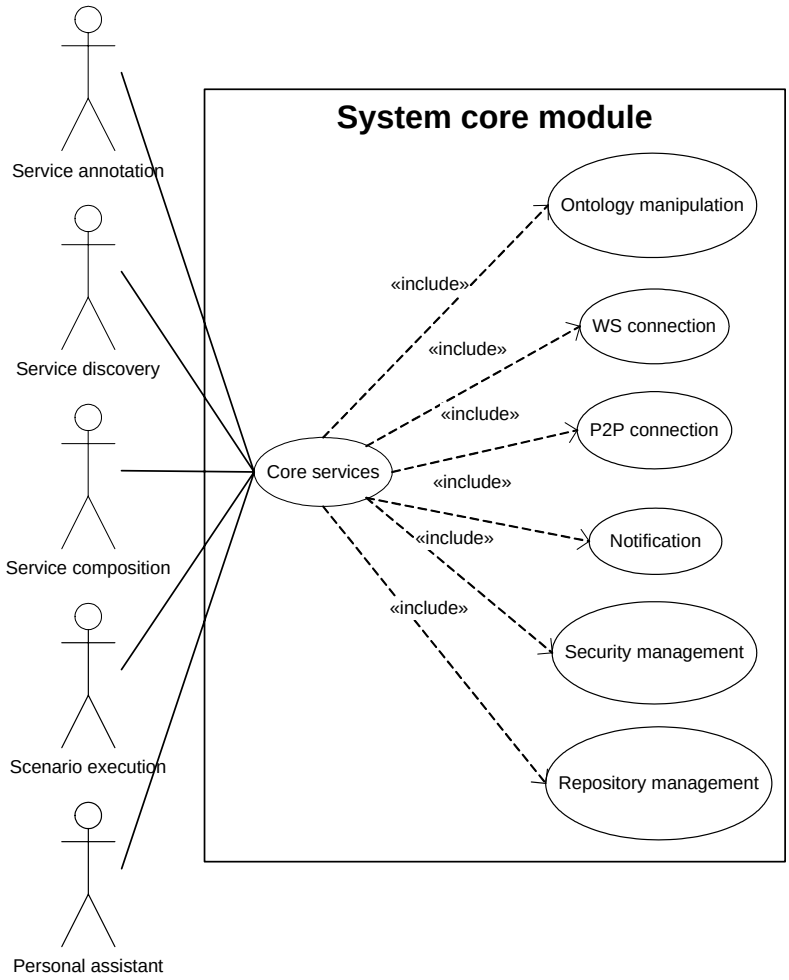

Figure 16 Use cases system core components

The description of actors follows:

- **Service annotator** Service annotator is an actor responsible for annotation of services with functional and non-functional properties. It uses directly ontology manipulation service (in case ontology has to be created, configured or deleted) and also services of repository management (loading and saving the descriptions of goals and services as well as the choreography interfaces of complex goals). The notification service is used in case the description of some goal was changed. The aim is to distribute notification of goal changes to those service providers whose services was earlier described according to this goal. By using the service of WS connection it can access and use the services of all platform components.
- **Service discovery.** Service discovery is responsible for discovery of services according to their properties. This actor uses repository management functionalities for loading the descriptions of goals, services, ontologies and mediators needed during the semantic discovery process of services. It uses also P2P connection manager functionalities (in case of accessing different P2P nodes) and also security management (in case the manipulation with confidential data is needed).
- **Service composition.** This actor is responsible for composition of services. It uses services of repository management for loading orchestration interfaces of complex goals.

- **Scenario execution.** This actor is responsible for executing services according to scenario. It uses repository management in case the process model needs to be saved or reloaded. The notification service is used in case a time out of a Web Service was exceeded. The security manager is used in case the confidential data are processed.
- **Personal assistant.** Personal assistant represents GUI for citizen and business users. By using the service of WS connection it can access and use the services of all platform components. Plus the service of repository management (in case of obtaining the catalogue of goals) and security management (in case the processing of personal data is required) are put to use by Personal Assistant.

The description of use cases follows:

- **Ontology manipulation.** In case the manipulation with ontology (or its part) is required. Manipulation with ontology consists of importing it into memory, deleting and updating. This use case is covered by the SWS ontology manipulation component (described in the next section).
- **WS connection.** In case Personal Assistant or Service annotator need to use the platform components the Access-eGov system provide them such opportunity. The WS connection manager component (described in the next section) provides such communication.
- **P2P connection.** If the incorporation of a different node is required in the discovery process, the Access-eGov system has to be able to access it. The P2P connection manager component (described in the next section) allows to use P2P technology for such a connection.
- **Notification.** The Notification service component (described in the next section) is responsible for the notifications distribution among the system. It runs e.g. when a time out of the service execution has to be distributed to the Personal assistant.
- **Security management.** If personal data or confidential data has to be processed the security management is needed. The bellow-described Security component is responsible for such security management.
- **Repository management.** When the manipulation with the repository data is required, such manipulation is handled by this core service. It enables reading and writing the repository.

10.2 Software components

The components defined in this module are:

1) SWS ontology manipulation component
2) WS connection manager component
3) P2P connection manager component
4) Data repository management
5) Security component
6) Notification service component

10.2.1 SWS ontology manipulation component

10.2.1.1 Component name and goal

The proposed short name for the **SWS O**ntology **M**anipulation Component is **SWSOM**.

The purpose of this component is to allow the manipulation with ontologies. Ontologies are saved in the repository (persistent layer). So, the SWSOM uses API object model for semantic web service ontologies and load the ontologies into these objects to the memory (API can be used also in personal assistant and annotation services).

10.2.1.2 Component API

There is only one API function for the SWS ontology manipulation component. It allows required manipulation with ontology by importing it into memory. The input to this API is identifier of an entity. The output is the identified entity - ontology (the aim of this API is to allow manipulation with ontologies).

Function	Description
Get entity (entity id): Entity	Load requested entity -ontology- from the repository to allow manipulation with it. Mostly, the service annotator will be accessing this function of the SWS ontology manipulation component, in case the necessary changes have to be performed, due to environment changes (e.g. a new law). **Inputs:** entity id - id of the entity – requested ontology is identified in the repository by a unique identifier (URI or IRI) **Outputs:** Entity – requested ontology is loaded into data memory object

10.2.2 WS connection manager component

10.2.2.1 Component name and goal

The proposed short name for the **WS C**onnection **M**anager Component is **WSCM**.

The goal of the WS connection manager software component is to create web service entry point which makes infrastructure components available to Access-eGov personal assistant and annotation services. The interface of this component consists of selected interfaces of the platform components (Discovery, Execution, and Composition modules). The main functionality of this component is to support marshalling and unmarshalling of parameters and return values from and to the web service protocol (which is used for communication between the AeG platform and clients).

10.2.2.2 Component API

WS connection manager exposes API of internal infrastructure components, i.e. Discovery, Execution and SWS ontology manipulation components.

10.2.3 P2P connection manager component

10.2.3.1 Component name and goal

The proposed short name for the **P2P C**onnection **M**anager Component is **P2PCM**.

The goal of the P2P connection manager software component is to create P2P entry point which makes infrastructure components available to other Access-eGov modules.

The distribution of request is useful during the semantic discovery process of services. This request distribution allows the services registered in different nodes of the Access-eGov infrastructure to be found and then also used during the execution.

10.2.3.2 Component API

There is only one API function for the P2P connection management component. In general, it allows the distribution of requests among the nodes in the overall infrastructure in the P2P network. The input to this API is a request (e.g. description of goal) and the output has a form relevant to this input (e.g. service description in case the request is a goal).

Function	Description
Request P2P repository/service	Request repository/service from remote nodes in the Access-eGov P2P infrastructure. If the distribution of these requests is needed it is managed by the P2P connection manager component thru this function. Other components do not handle with any aspects of such management and access only. **Inputs:** Repository/service request that have to be distributed among Access-eGov infrastructure. Most likely only one request will be distributed by one call of this function. **Outputs:** Requested output from the different nodes than the node which initiated the distribution. There might be a lot outputs – so the format of the output of this function should be a List

10.2.4 Data repository component

10.2.4.1 Component name and goal

The proposed short name for the **D**ata **R**epository Component is **DR**.

The goal of the Data repository software component is to create entry point (interface) which makes data repositories in persistence layer available to Access-eGov infrastructure components.

Therefore, this component has to provide functions, which allows manipulations with all entities, i.e. with goals, services, ontologies, mediators, orchestration interfaces and profiles. Where the manipulation means insertion of a new entity, updating of an existing entity, deletion of an existing entity and selection of an entity.

10.2.4.2 Component API

There are four API functions provided by Data repository component. Their functionalities allow insertion, updating, deletion and selection of entities. The input to this API are entities for insertion, updating (entity with their repository ID) and deletion (entity their repository ID), or query in case of selection. The outputs are inserted entity (entity and their new ID in the repository), updated entity, deleted entity and a list of selected entities respectively.

Function	Description
Insert(Entity): Entity	Insert entity into appropriate data repository in the persistence layer according specific entity type in case a new entity emerges and it has to be stored. It involves also storage of the current process states during the execution, but of course this function will be used in case a new service (its description), new goal, new ontology, new mediator or new user profile will be created. It is expected that this situation will be arising quite often in the AeG system. **Inputs:** Entity to insert – all entity kinds mentioned in the previous description which have to be inserted into date repository (in the AeG persistent layer). **Outputs:** Inserted entity – entity with the 'flag' giving a signal that the entity was stored successfully into data storage in the persistent layer can be its ID (e.g. IRI).
Update(Entity): Entity	Update entity in appropriate data repository in the persistence layer. This function will be used in case an already existing entity has been loaded from the repository and the necessary changes were done on it. Then the update of the old state in the repository is needed and the handling with this update is up to this function of the Data repository component. It is expected that this situation will be arising quite or very often in the AeG system. **Inputs:** Entity to update – entity, which was changed in the memory and these changes have to be updated in the repository. **Outputs:** Updated entity - entity with the 'flag' giving a signal that the entity was updated successfully.
Delete(Entity): Entity	Delete entity from the appropriate data repository in the persistence layer. This function will be used in case the entity in the repository will be not needed anymore. It involves mainly old service and goal descriptions and states of the execution processes which will be not used anymore (e.g. user cancels the process). Note, that the decision whether some entity will be deleted does not depend on the Data repository

	component or even on this function. It is expected that this situation will be arising quite often in the AeG system. **Inputs:** Entity to delete – The entity which can not be used anymore (e.g. because new entity replaces their function, or this entity is not actual etc.) so there is no reason to leave it in the storage in the AeG persistent layer any longer. **Outputs:** Deleted entity - entity with the 'flag' giving a signal that the entity was deleted successfully, so there is no such entity in he storage in the AeG persistent layer.
Select(Query): Entity	Select entities from appropriate data repository according specific entity type and query request. This function will be mainly used in case the searching process for a specific entity is being performed in the AeG system. The query should address the type of entity for selection. It is expected that this situation will be arising very often in the AeG system. **Inputs:** Query request – most likely a request which arises during the process of service discovery. The query is matched against the descriptions of goals and services. **Outputs:** List of selected entities – entities which are appropriate to the query are selected and returned a an output from this function.

10.2.5 Security component

10.2.5.1 Component name and goal

The proposed short name for the **Sec**urity Component is **SEC**.

Security software component checks user credentials (together with information about the user's profile and the security scheme as required by a public agency's service) and issues security tokens that public agencies may use as temporary Single-Sign-On authentication.

Security in Access-eGov is not one-dimensional but divided into two equally important parts. Communication Security involves securing all communication channels and also securing a correct identification of communication partners. Basic access control with only the ability to allow or deny access based on used protocol or IP of the client also falls into this category. While these requirements are very important, their implementation is simple. There are encrypted versions or encryption add-ons to all of the standard communication protocols Access-eGov uses.

The challenging part of the Access-eGov security architecture is the system security. Access-eGov is in the process of developing a novel system architecture revolving around the notion of privacy preferences. With privacy preferences, users can specify which of their personal data can be transmitted to the platform and under which circumstances. A user for example could accept sharing of his credit card number only through encrypted channels and only if

the receiver is a financial institution. A number of technical implementation models are currently researched, that could further strengthen privacy and flexibility by placing central components of the security component in a SOA themselves which in turn allows the use of the same semantic components to string together a chain of security services.

10.2.5.2 Component API

Component API will be specified in the subsequent project reports (D4.1, D4.2 and output of T5.3). The exported functionality of this component however, will be rather simple.

Function	Description
Authorize(User, Resource): AuthorizationToken	Evaluates if a particular user is allowed to access a given resource **Inputs:** User model, containing all necessary information including privacy preferences, Resource to which access is sought **Outputs:** AuthorizationToken

10.2.6 Notification service component

10.2.6.1 Component name and goal

The proposed short name for the **N**otification **S**ervice Component is **NS**.

Notification service component distributes internal messages required to coordinate activities of Access-eGov Infrastructure components and notifications to clients via connection manager web service interface (examples of the messages include notification when some data structures are changed or timeout notifications generated during the execution of scenario processes). Each AeG component should be able to use internal message delivery service to achieve asynchronous communication inside the AeG infrastructure. This offers the possibility to adopt requirements to modify the execution semantics of the AeG system that could later arise. Flexible asynchronous communication supposes the temporal and referential decoupling of the send and receives operations. One of the possibilities how to achieve such a decoupling is to use a communication mechanism based on shared space.

Beside the asynchronous communication, the NS component offers also the notification service. In the process of notification we distinguish producers (event generators) and consumers (listeners). Events are generated by producers and the system components can register interest to be informed on specific events, whereby they specify also the manner, how to react on the event. As the Access-eGov system is a distributed system based on peer-to-peer infrastructure, the event delivery system has to be also distributed.

10.2.6.2 Component API

Function	Description
Write(message, Shared Space): boolean	Write a message to the shared space, where it will be available to other AeG components. This operation together with read separates temporally and referentially the communication processes and enables asynchronous communication. **Inputs:** message – set of serialised objects that

	represent data to be send from an AeG component to the potential receivers inside the AeG infrastructure; Shared Space – distributed blackboard mechanism accessible for all AeG components **Outputs:** True in case of successful write operation into the shared space, otherwise returns false
Read(template, Shared Space): message	Read a message from the shared space according a template message. This operation together with the write separates temporally and referentially the communication processes and enables asynchronous communication. **Inputs:** template – set of serialised objects that represent template data to be read from the shared space (if the data are not available, the caller is blocked until such information occurs in the shared space); Shared Space – distributed blackboard mechanism accessible for all AeG components **Outputs:** message - set of serialised objects that composes a message in the shared space and corresponds to the defined template.
Notify(Shared Space, template, Listener): boolean	Register a consumer (Listener) for handling of a specific event (Event) generated by the producer (EventGenerator). In case that the producer creates an event, all listeners (event consumers) are notified via distributed event delivery mechanism about occurred event. **Inputs:** Shared Space – distributed blackboard mechanism accessible for all AeG component; template – set of serialised objects that represent template data, which writing into the shared space generates the notification; Listener – object specified by the consumer interested into the message occurrence specified by the template **Outputs:** true in case of successful register operation, otherwise returns false
Register(EventGenerator, Event, Listener): boolean	Register a consumer (Listener) for handling of a specific event (Event) generated by the producer (EventGenerator). In case that the producer creates an event, all listeners (event consumers) are notified via distributed event delivery mechanism about occurred event. **Inputs:** EventGenerator – object that represent the event generator; Event – event that can be produced by EventGenerator and that is interesting to the consumer; Listener – object specified by the consumer interested into the Event, that should be notified about

	the Event occurrence **Outputs:** true in case of successful register operation, otherwise returns false

10.3 Software components sequence diagrams

Interactions with the System module components are depicted in the sequence diagrams of the referenced modules.

11 Conclusions

In this document, we outlined the detailed functional description of platform components, which were identified to fulfil the user requirements followed from the related analysis phase. Based on the results of the user requirements analysis (described in D2.2) and the overall architecture of the Access-eGov system (described in D3.1) we identified required software components with required set of functions and grouped them logically into software modules. Each module is responsible for specific functional task within the Access-eGov architecture.

We described the components at the logical level in terms of their names, functionality and application programming interface, but with sufficient details to build modules related to users requirements. Modules are then described also at the logical level in terms of names, included components and their interactions. We used UML models (use case diagrams and sequence diagrams) to describe the functionality and interactions of each module in a common semi-formal way.

This specification will be used for further development (design and implementation) phases. Now we have a clear understanding about functionality of the system and the details about each software components. The following future software development tasks of this project continue with the development of the Access-eGov system:

T4.1 – According to the presented system architecture and functional description on the logical level the structure and the interface each software component is designed. The design must be in the sense of selected implementation technology.

T4.2 – Detailed specification and design of the security issues for the Access-eGov system functionality.

T5.1 – Design of specified personal assistant software components (structure and interface).

T5.3 – Detailed specification and design of the proposed security components in this report.

T7.2 – Development of a guideline for the semantic mark-up framework and the service annotation process.

T7.3 – Design and implementation of the semantic mark-up framework and the service annotation process.

Annex A: Used Technologies

Since the WSMX platform currently is not widely used in industrial projects, stable figures describing its real-world performance do not exist. Its use in several research projects[1] so far is not targeted at performance, but functionality. Early industrial adopters, like NIWA or Sybernet, did not provide benchmark data yet. Therefore we have to assess the performance issues with regards to the WSMX platform.

Potential performance bottlenecks exist above all in the data repositories and the semantic components. Communications and Resource Manager and Parser components are straightforward components with a linear complexity. The data repositories and parts of the semantic components on the other hand are the most complex.

Access-eGov will use Object-relational-Mapping (ORM) concepts for storing the Data Repositories' data which are quite common and proven nowadays. The most likely choice for us, hibernate, has a large user-base with quite demanding applications and web sites. Many of those users report great performance and scalability on relevant forums. Official benchmarks do not exist for hibernate for a number of reasons[2].

The letter sent on March 17, 2007 from the DERI expert Thomas Haselwanter describes actual situation of benchmarks and performance tests for WSMX:

"*...nobody has made any benchmarks for WSMX yet, nor am I aware of any plans for something like this. Frankly, nobody was really interested in this in any of the several EU projects where we used this platform, since everybody only cared about the functional requirements that WSMX could fulfil. Performance issues were always considered to be part of a much later stage when the results of this research would be commercialised at some point. With that being said and out of the way, my gut feeling tells me that the single most critical hotspot in terms of performance is the reasoner since it's used extensively inside practically every component of WSMX. We've been unsatisfied with the performance (among other things) of the reasoners we've been using up to now, so we're implementing our own reasoner called IRIS, of which the first version has recently been released. Again no benchmarks yet, but when benchmarks appear they will probably appear for the reasoning subsystem rather than for all of WSMX, because the measurement that can be taken here are more meaningful since pure reasoning doesn't involve a lot of side effects (network and others) that WSMX is exposed to, while still playing a key role inside WSMX to have a real effect on the overall performance...*"

Other components needed for Matching or Discovery also have a linear complexity apart from the Reasoner. Even DERI, the creators of WSMX, can see the performance problems in only the Reasoner component as they conveyed in email discussions. And the Reasoner currently used for WSMX indeed had performance problems, which lead DERI to implement their own Reasoner[3].

[1] DIP, SEKT, Knowledge Web, Cocoon, SUPER, SWING, Tripcom, SemanticGov, SAOR, KMI

[2] http://www.hibernate.org/157.html

[3] http://iris.deri.at

A.1 Benchmarks and performance tests

Having analysed the AeG architecture, we have concluded that the performance of the whole AeG platform is mainly dependent on the Discovery module. Components of this module, namely Matching and Filtering, are invoked either explicitly from the Personal Assistant client to find services, or from the Execution module to resolve unresolved sub-goals during the scenario execution. The performance of the Matching and Filtering components depends on the implementation of the Reasoning component and on the implementation of the Service and Ontology repositories. Generally, we can say that from the performance and scalability point of view, highly scalable and efficient Reasoning component is crucial for all semantic based applications; not only for the AeG platform.

It should be noted that implementation of an efficient reasoner covering all the variants of the Web Service Modelling Language is out of the scope of the AeG project. For this reason we will reuse the existing implementations. The survey on existing reasoners, which covers requirements for various WSML variants, can be found in [D16.02 WSML Reasoning Implementation]. The published benchmarks performed on these reasoners are based on general settings and test mainly querying on large instance sets and atomic operations like subsumption of concepts. But in the context of the AeG platform (and the semantic services generally), Reasoner is used to compute a degree of matching between functional and non-functional properties of goals and services specified as the logical expressions.

This application of the reasoner in the AeG system, where the terms of one logical expression defined for the goal are matched to the terms of another one defined for the service (WSMO conceptualisation), represent a more complex problem than the problems measured by the existing benchmarks. Therefore, the results of the existing benchmarks (corresponding rather to “atomic” operations) may not be fully relevant for the AeG implementation.

Another important issues, which can influence the performance measuring, are related to constraining the expressivity of the logical language for the particular domain and to the fact that the matching implicitly takes into consideration not only the logical inference, but also mediation (i.e. if an instance of a concept from one ontology can be transformed to instance(s) of another one).

Note, that within the scope of the AeG pilot applications, operational conditions with a limited number of services and goals will not allow to estimate high load performance of the AeG system. For these reasons we have designed our own benchmarks and we will carry out AeG performance tests in the following way:

- To analyse the domain ontologies and identify common design patterns which will be used to constrain expressivity of the logical language for testing.
- To implement generator used to generate constrained logical expressions for functional and non-functional properties of goals and service profiles.
- To generate a testing set of the services and goals (this includes generation of referenced domain ontologies).
- To test a composition of the components for semantic matching (i.e. Matching component, Reasoning component, Mediation component and Filtering Component without the Data Repository access) with randomly selected goals and with a different number of registered services to estimate performance and scalability of reasoning and mediation. These tests should cover both "simple" semantic matching and "rich" semantic matching with randomly generated inputs (see chapter 7.2.2).

- To test the whole functionality of the Discovery module, i.e. Matching component together with the access to the service repository.

In this way, we can precisely control operational conditions and use artificial data to estimate the performance and scalability of the platform for high load applications with hundreds of goals and thousands of registered services.

A.2 Scalability and simultaneous access

Besides of the performance issues related to the Discovery module, we have additionally identified that the scalability of the whole system is mainly influenced by the scalability of the Service and Ontology repositories and the scalability of the Goal and Scenario Execution component. Scalability of the repositories will be directly tested within the scope of the performance tests described in the previous chapter (i.e. we can measure storage scalability with storing and querying on a generated large set of services which can refer to a large set of concepts from artificial domain).

The implementation of the Goal and Scenario Execution component is mainly important from the point of view of simultaneous access. It has to be noted that in the context of the AeG project with hybrid scenarios, "real time" invocation of the electronic services will be mixed with discontinuous long-lasting transactions with the traditional services. It means that the process of the execution will be frequently suspended and the system will wait for the user interaction (so, for instance, it is important to take into consideration not only the efficient interpretation of the process activities but also the updating and restoring of the process execution).

Since it is planned that this component will be implemented from scratch and will be not based on the WSMX components, we have designed our own scalability tests, similarly as for the Discovery, in the following way:

- To analyze workflow and dataflow patterns used in the hybrid process models of the life event scenarios.
- To implement generator which will generate process models with a various configuration of the activities (i.e. branching, looping, concurrent execution, invocation of the web services, and interaction with the traditional services) according to the identified patterns.
- To simulate concurrent execution of randomly selected processes and measure the scalability of the number of concurrent processes.

To conclude, our strategy for measuring the scalability and performance of the whole AeG system is summarized in the table below.

Component	Tested	Targeted issues
Matching, Filtering	Directly (on a generated set of services and goals)	Scalability depending on the number of services, number of ontology concepts, matching efficiency, concurrent access.
Reasoning	Indirectly through Matching and Goal and Scenario Execution	Matching efficiency

Resolving, Chaining	Indirectly through Matching	
Data repository	Directly (Service and Ontology repositories on the generated set of the services and goals)	Scalability depending on the number of services, number of ontology concepts, concurrent access
P2P Connection Manager	Indirectly through Data repository	
Goal and Scenario Execution	Directly (simulation with generated processes)	Scalability depending on the number of users, concurrent access, reasoning efficiency (indirectly - querying of the process states)

References

[1] D3.1 Access-eGov Platform Architecture

[2] D2.2 User requirement analysis & development/test recommendations

Glossary

The following terms are used within this report. In case the term was defined in another Access-eGov deliverable the reference to the source is included.

Access-eGov annotation services: A web-based application that is not an integral part of the Access-eGov infrastructure. Its main purpose is to allow domain experts to semantically describe their electronic/traditional services using their respective public service ontology. This will explicitly involve annotating traditional web sites as well. (D3.1, p. 44)

Annotated service: Semantically described service, i.e. service extended with references to some semantic models, where semantic model means within the Access-eGov project ontology or concept in the ontology.

Architecture: The overall design or structure of a software application

Chaining component (CHAIN): CHAIN composes services by recursively regarding the effects and outputs of one service as the preconditions and inputs of a following one until a desired effect is reached. (D3.2, p. 28)

Complex goal: An internal process model which consists of abstract activities (goals). (D3.2, p. 27)

Composed services: This term specifies composed activities which fulfil goals in the generic scenarios. The term activity usually means either electronic service or traditional service (an atomic activity). But in case the goal from generic scenario can be resolved only during the time of execution (some inputs depends on some of the previous obtained outputs), the activity has an abstract nature. (D3.1 p. 36)

Composition: A process of orchestration of the existing services to the new scenario to solve complex goal. (D3.2, p. 24)

Data repositories: Data repositories store user various data objects used by other components, where various data objects means life events/goals, service description, ontologies, process context and security data. (D3.1, p. 44)

Discovery: A process of searching goals and services for the process model composition. (D3.2, p. 16)

Electronic service: A service which is performed electronically.

Execution: A process of execution of the retrieved service or workflow (composed service). (D3.2, p. 30)

Filtering component (FIL): FIL uses non-functional properties to additionally filter or reorder discovered services according to the requester preferences. (D3.2, p. 19)

Full-text search component (FTS): FTS provides an interface to the full-text search of services and life events/goals.

Functional properties of goals and services: These properties describe inputs, outputs, preconditions and effects of the service (IOPEs). (D3.1. p. 36)

General descriptions: General descriptions are properties which could include contact details for persons that are responsible for the service's operations and could be part of the non-functional properties. (D3.1, p. 22)

Goal: A goal specifies those objectives that a client might have when consulting a service, including functionalities that a service should provide from the user's perspective. (D3.1, p. 35)

Information consumer related module (MC): A module which specifies a goal and executes the retrieved services or a module which allows general access management of the Access-eGov platform services. (D3.2, p. 10)

Information provider related module (MP): A module which copes with the main tasks of information providers (annotating/registering services and building goals and complex goals). (D3.2, p. 10)

Life event: A life event denotes a specific situation (i.e. event) in the life of a citizen or a life cycle of an organization that requires a set of public services to be performed. (D3.1, p. 34)

Life events and goals management component (LEGM): LEGM provides the functionality for description of atomic (simple) goals and for composition of these goals to the more complex generic scenarios to specify life events. (D3.2, p. 14)

Life events/goals repository: Manages goals and generic scenarios associated with the life events. (D3.1, p. 44)

Matching component (MAT): MAT provides matching based on the outputs and effects or matching based on rich semantics. (D3.2, p. 18)

Mediation component (MED): The role of the MED is to reconcile the semantic and data heterogeneity that can appear during discovery, composition or execution. (D3.2, p. 20)

Mediation: A process of reconciliation of the semantic and data heterogeneity that can appear during discovery, composition or execution. (D3.2, p. 20)

Non-functional properties: These properties describe the semi-structured information intended for the requesters for service discovery, e.g. service name, description etc. (D3.1. p. 36)

Notification services component (NS): NS distributes internal messages required to coordinate activities of Access-eGov infrastructure components and notifications to clients via connection manager web service interface.

Ontology repository: This repository stores domain ontologies and associated mappings for mediators. (D3.1, p. 44)

P2P connection manager component (P2PCM): P2PCM creates P2P entry point which makes infrastructure components available to other Access-eGov modules. (D3.2, p. 47)

Persistence layer: provides interfaces to store and retrieve various data objects used by other components. (D3.1, p. 44)

Personal assistant client: A component which manages the user profile repository and it may additionally manage repositories intended for the caching of other data objects like life events and goals in order to improve user interface responses. (D3.1, p. 45)

Public administration tools: Tools which are needed by public administration institution to annotate their services, build goals as well as complex goals.

Public agencies: Agencies which either provide their customers with public administration services or mediate between them and public administration institution.

Reasoning component (REAS): REAS allows to explore the domain knowledge about the input and output types. (D3.2, p. 20)

Resolving component (RES): RES resolves abstract activities (sub-goals) into services by using Discovery module. (D3.2, p. 26)

Security component (SEC): SEC checks user credentials and issues security tokens. (D3.2, p. 31)

Security scheme repository: This repository stores user login information and access rights. (D3.1, p. 44)

Security surroundings: This surroundings includes information who is eligible to actually use that service, what form of identification is required concerning the privacy policies that the service itself can offer. (D3.1, p. 49)

Semantic matching: A matching strategy which considers meaning of matched elements (e.g. functional or non-functional properties of services).

Service annotation component (SA): SA creates service profile, which consists of functional and non-functional properties. (D3.2, p. 12)

Service Profile: A Service profile specifies what does the service provides from user perspective and is used by the public administration to advertise services. Service profile consists of non-functional and functional properties. (D3.1. p. 36)

Service repository: This repository stores descriptions of web services, traditional services and composed services registered for specific orchestrated scenarios. (D3.1, p. 44)

Software component: A software component is a piece of software that performs a specific technical task. Software components are grouped in software modules. (D3.2, p. 7)

Software module: A software module is a piece of software that performs a specific functional task. (D3.2, p. 8)

SWS ontology manipulation component (SWSOM): SWSOM is in-memory object model for semantic web services ontologies. (D3.2, p. 46)

System core component: A component which is responsible for the interaction with the user of the Access-eGov platform and also for tasks, which are relevant to the core platform functionality. (D3.2, p. 43)

System management related module (MM): A module which cope with the core functionality of the Access-eGov platform. (D3.2, p. 10)

Traditional service: A service which is performed without the electronic support.

Visualisation and Data Entry component (VN): VN is responsible for visualisation and interaction during the whole process of scenario execution. (D3.2, p. 40)

WS connection manager component (WSCM): WSCM creates web service entry point which makes infrastructure components available to Access-eGov personal assistant and annotation services. (D3.2, p. 46)