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

D3.1 Access-eGov Platform Architecture

Start date of project: January 1, 2006 **Duration:** 36 months
Date of submission: February 12, 2007
Lead contractor for this deliverable: University of Regensburg
Revision: Final 1.1
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

D3.1 Access-eGov Platform Architecture

Workpackage:	WP3	**Task:**	T3.2

Date of submission:	19 February 2007
Lead contractor for this deliverable:	University of Regensburg (UR)
Authors:	Peter Bednar (TUKE) Stefan Duerbeck (UR) Jan Hreno (TUKE) Marian Mach (TUKE) Lukasz Ryfa (EMA) Rolf Schillinger (UR)
Version:	1.1
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), Slovakia; City Hall of Gliwice (GLI), Poland; State Government of Schleswig-Holstein (SHG), Germany.

Abstract:

This report outlines the overall Access-eGov system architecture. It provides an overview of the conceptual, logical and physical architecture from different points of view: security-focused, functional, structural and data-based perspectives.

The report provides an overview of all the pilot implementations and user requirements that the system has to live up to. These requirements are dealt with on the conceptual architecture level, by describing the system interaction with the user(s). In addition, a number of possible software architectures are researched for use in the Access-eGov. An architecture on the logical level is proposed, taking into consideration the researched architectures. This architecture proposal is followed by an outline of some details of the envisioned physical level and its possible implementation. Finally, the report analyzes some of the major software risks that are assumed to pose a threat to the overall project success.

Document Sign-off

Nature of Sign-off (Reviewed/Approved/Submitted)	Name	Role	Participant short name	Date
Reviewed	Rolf Schillinger	TL	UR	February 16, 2007
Approved	Martin Tomasek	WPL	IS	February 19, 2007
Submitted	Tomas Sabol	PM	TUK	February 19, 2007

Document Change Record

Date (d/m/y)	Version	Contributor(s)	Change Details
17/07/2006	0.01	Rolf Schillinger	Preliminary report outline
22/08/2006	0.05	Rolf Schillinger Stefan Dürbeck	Added content (mainly requirements and possible architectures)
30/08/2006	0.07	Peter Bednar Jan Hreno	Added content (mainly structure and knowledge formalisms)
5/09/2006	0.08	Marian Mach	Report restructuring, chapter 1 extended
18/09/2006	0.09	Rolf Schillinger Stefan Dürbeck	major changes to the overall document structure
21/09/2006	0.10	Peter Bednar	major changes to the overall document structure
22/09/2006	0.11	Rolf Schillinger Stefan Dürbeck	Comments incorporation and content correction
25/09/2006	0.12	Stefan Ukena	Section 3.3.5 about AeG ontologies
27/09/2006	0.14	Peter Bednar	Added 3.3, changes to chapter 5
28/09/2006	0.15	Rolf Schillinger, Stefan Dürbeck	Corrections in language and writing style throughout the document
28/09/2006	0.16	Lukasz Ryfa	Supplements to section 4.5.8
29/09/2006	1.0	Rolf Schillinger, Stefan Duerbeck	Final version
29/09/2006	1.0.1	Marian Mach	Polishing of the final version
10/02/2007	1.1	Marek Skokan	Reformatting (footnotes and pictures in Annex I)
15/02/2007	1.1	Stefan Dürbeck	Executive summary added

Files

Software Products	User files / URL
Microsoft WORD	

Content

Table of figures

Executive Summary

This architectural report outlines our ideas of how the Access-eGov platform is going to be built regarding its internal module structure. Thus, it represents a linking document between the functional birds-eye view of the system as given in the Technical Annex, and the description of the respective components' functionality of upcoming deliverable D3.2 (Access-eGov components functional description). The aim was to break the system up into architectural pieces (viz. modules) and to abstractly define their boundaries and mutual dependencies.
This was achieved applying a top-down approach emanating from a set of high-level use cases based on user requirements and pilot scenarios. Two classes of system users could be identified using a user-centric approach to conceptual architecture:

- Information Providers (e.g. Public Servants), charged with service-annotation, -registration and workflow-definition;
- Information Consumers (e.g. Citizens, Businesses, other Public Service Agencies), willing to specify their goals towards the system, triggering and executing the services found.

The logical architecture analysis focused on user-machine interaction and resulted in isolating main functionality blocks of the Access-eGov architecture that will take charge of:

- user input via the Personal Assistant Client;
- Annotation, Discovery, Composition, Execution of services;
- Mediation between different semantic annotation techniques;
- Persistence of service- and ontology-related data.

The Physical Architecture analysis resulted in selecting the frameworks that will be used and tailored to satisfy our system requirements. The report concludes that Access-eGov will handle data structures using a service-oriented architecture approach. WSMX and relating technologies are the framework of choice to provide the basement for the Access-eGov platform. The platform will constitute a “small world” network with participating nodes interconnecting in a peer-to-peer method of operation.
In order to justify our architectural decision taken in this report, we recapitulated some technological aspects that have already been mentioned in D2.2 State-of-the-art analysis.

The assessment of risks that may arise from architectural decisions taken in this report, concluded that even threats resulting from the project-specific environment are unlikely to occur. The architectural decisions are expected to have no disadvantageous impact on the overall-success of Access-eGov.

This report has to be regarded as first in a series of written definitions of the platform structure and of the interaction between functional modules. Taking a birds-eye view, it defines system boundaries for the upcoming reports D3.2 (Access-eGov components functional description), D4.1 (Specification of components for “Mark-up services”) etc. and D5.1 (Specification of components for “Personal Assistant Client”) etc. which will verbally describe the detailed functionality of important system components.

1 Introduction

As this is rather challenging and novel task, the general architecture of the Access-eGov system has to be carefully designed. It will be based on concepts and methods proposed and implemented in various fields of research. The project can only in limited terms rely on already existing de-facto or industry standards. At the time of writing the report, there are hardly any operating implementations of information systems using semantic technologies, and some best-practise solutions are not available.

Firstly, this document provides an overview of the pilot implementations and user requirements that the system has to live up to. These requirements are dealt with in chapter on the conceptual architecture, describing the system interaction with the user(s). In addition, a number of possible software architectures are researched for use in Access-eGov prior to outlining a first logical architecture proposal, considering the researched architectures. Following this, the next chapter focuses on the overall design and on certain details of the envisioned physical architecture, and the possible implementation.

1.1 Objectives and scope

The objective of this report is to define a high level software architecture that is needed to deliver services for both service providers (i.e. public authorities) as well as service consumers (citizens and/or businesses).

Therefore, the report covers several areas while subsequently building upon and extending work that has previously been done within Access-eGov. In particular, our architectural design is influenced by two main sources:

- Requirements on the architecture to be designed (based on the content of the Access-eGov Deliverable D2.2 [D2.2]), and
- Information on technology trends relevant for the delivery of services (based on the Access-eGov Deliverable D2.1 [D2.1]).

The report deals with the architecture of the supporting infrastructure that needs to be realized to achieve easy location of services, their execution and their interoperability. The architecture shall be designed in a way that allows its use on local, municipal, regional or cross country administrative level.

An ‘architectural view’ of systems (both of business- and IT-systems) is defined in the ANSI/IEEE Standard 1471-2000 as “the fundamental organisation of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution”. But in practice, the term ‘architecture’ can have a slightly different meaning for different institutions and organisations. In order to avoid confusion, the following lists are supposed to define the scope of the term ‘architecture’ for our common understanding.

The architectural design is expected to **consist of**:
- Non-functional requirements,

- Functional scope and responsibilities of the architecture,
- High level design,
- High level functional specifications of architectural components,
- Design constraints.

Architectural design will **not deliver**:

- Comprehensive functional requirements of architecture elements (since these will be delivered by the Access-eGov Deliverable D3.2),
- Detailed data and metadata analysis (which is expected to be targeted within the Access-eGov workpackage WP7),
- Low-level information on implementation issues (which will be solved within the Access-eGov workpackages WP4 and WP5).

1.2 Approach

The architectural description (as well as the document structure) is based on a matrix approach combining four views and four abstraction levels in order to compose a coherent description framework. The framework is depicted in Figure 1.

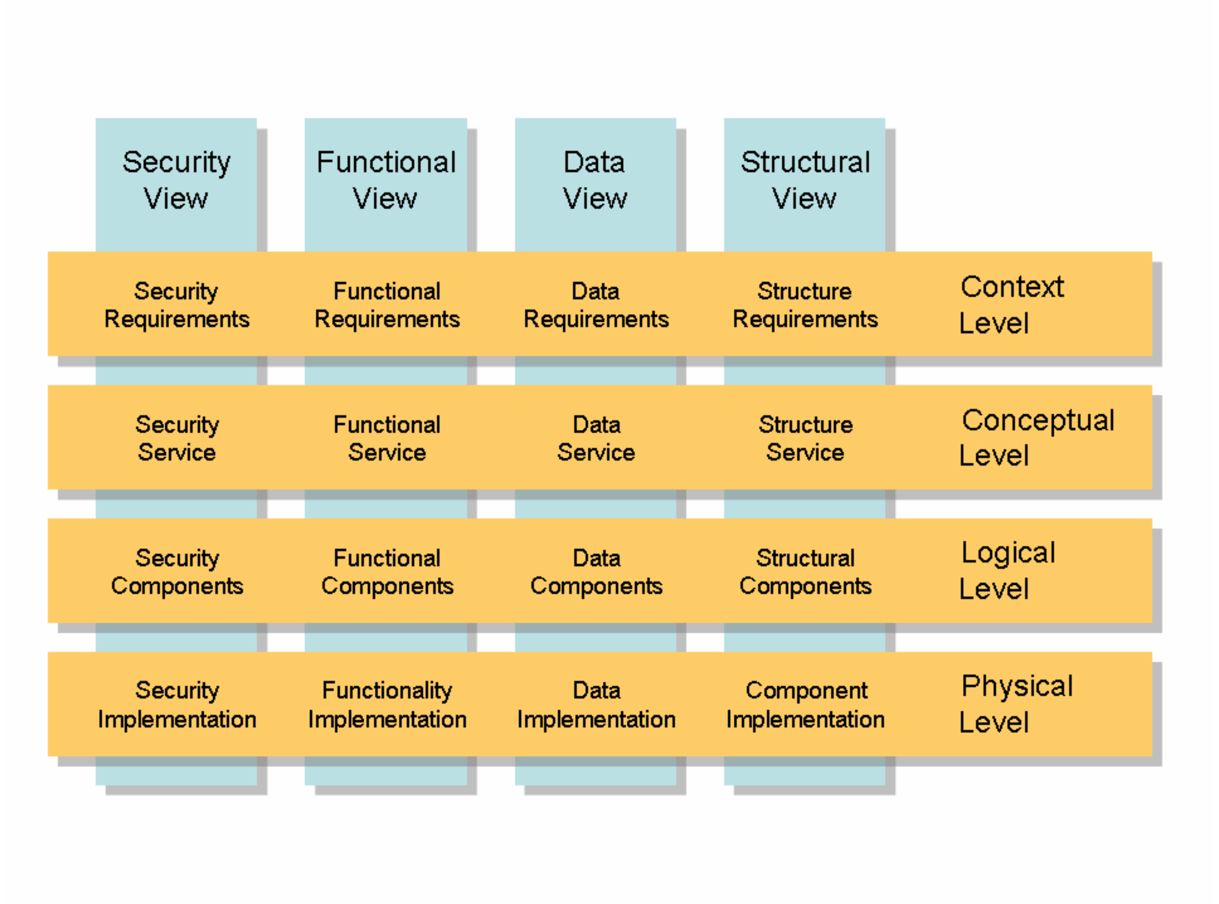

Figure 1 Architecture description framework

This framework addresses four distinct views: Security, Functionality, Data (as well as metadata), and Structure. There are strong interdependencies between them, since they represent complementary views. For example, the structural view can understand the architecture in terms of the modules that the whole system is decomposed into, while the data view can focus its attention on data and information exchanged by these modules. The security view is dedicated to the functionality relevant to security issues, while the functional view focuses on domain-specific functionality provided to users. Ideally, all complementary views have to be incorporated into the architecture design.

Furthermore, the framework recognizes four levels of abstraction: Context, Conceptual, Logical, and Physical level.

- The context level is supposed to provide answers to the "*why*" question and to provide context information and key principles supporting the value proposition for the architecture to be developed. It defines the principles, vision and scope that will rule the new architecture design. A comprehensive set of requirements is used in a consistent manner in the decision making process.

- The conceptual level addresses the "*what*" aspect of architecture design. It defines the services that are required and what is required for each service. It specifies the services and interactions between these services in support of the principles defined in the upper level.

- The logical level derives solutions from the conceptual level of "*how*" to meet customer needs, showing how the components interrelate and where the components implement their services. It describes the solution as a technology-neutral set of components, including a clear definition of the integration and collaboration contracts between these components. It remains independent of particular products.

- The physical level (i.e. the lowest and most technical level) addresses the "*with what*" aspects of architecture design and defines applicable standards, products, tools, frameworks and components, package solutions, etc. for further development and implementation.

At some level, typically at logical and/or physical levels, there can usually be more than one way to approach a solution (which reflects various drivers, e.g. costs, flexibility, security, manageability, etc.). The key decision that will have to be made at this level is to select the most adequate solution alternative that delivers the required services, and best addresses the guiding principles.

The approach presented above constitutes a pragmatic way to consider the more relevant framework parts, in comparison to other framework models that are suggesting treating all architecture aspects as equally important.

2 Architecture context

2.1 Pilots

Accompanying the research and development, and to finally verify the results of Access-eGov, several pilot scenarios have been established and their specific requirements have been surveyed (cf. chapter 2.3.2 in this document). While it is desirable to fulfil as many user requirements as possible in these pilots, the researcher-developers will need to seek a compromise with the user partners, not to implement customized software for research purposes only, but to deliver a usable product that fits both, the research goals of this project and the wishes of the user partners. The next paragraphs contain the summaries of the pilot scenarios taken from the "User requirements Analysis" [D2.2]. The Pilot scenarios are outlined prior to describing the overall architecture, because they constitute the source of all development and research work that needs to be undertaken by the project partners.

2.1.1 Slovak pilot

This activity scenario is based on the intention of building a new family house in a municipality of the Košice region. At present, it is difficult for the average citizen to make his/her way through the trap doors of complex administrative processes while obtaining a building permission. The Access-eGov system is intended to facilitate such procedures using an interactive web platform which provides citizens with useful guidance on "what and how to do it". As a result, a user shall be easily guided through all of the "building permission procedures" required, by avoiding unnecessary additional questions or procedures.

The value added by the Access-eGov solution consists of an increased efficiency and performance that is achieved through process optimization and integration of the concerned public services at execution time. All this adds up to greater convenience for the citizen as final beneficiary. This will also ensure more transparency in the public service delivery and shall encourage people to make use of electronic public services more often. The ICT components to be used by the Slovak pilot are:

- Electronic correspondence
- Online forms provision
- Online tracking of the procedure (graphic indication of current status, timings and countdown of stated time period etc.)
- Online information about the costs of procedure (parts)
- Estimation of project documentation costs
- Online list of all relevant institutions

2.1.2 Polish pilot

The service described in the scenario is the establishment of an enterprise (starting one's own business) by the user. This service consists of four main tasks:

- Registration in the City Hall (the local government).
- Registration in the Statistical Office.
- Registration in the Tax Office.
- Registration in the Social Insurance Agency.

The main goal of delivering that service is to enable citizens to establish their enterprise via the Internet (in those cases where it is possible) and to deliver complete information package related to the service. The information will be provided by dialogues between the user and an intelligent agent (a component of the Access-eGov platform), correct interpretation of the user’s queries, and additional questions to the user. The aim of performing each task is to give the user all required instructions, to point out activities he should perform, places/institutions he should go to, forms he should fill in, and to provide access to e-services. The overall goal is to support the user to start his business.

The scenario is based on the general description of the Access-eGov solution as an IT system supporting citizens or businesses in the context of public service provision. In other words, the Access-eGov platform will use the detailed and semantically annotated information about the public services in order to provide the customer with appropriate advice on steps which have to be taken in a particular business episode or life-event. The system should act as a CRM system with a profile of the user in order to suggest an appropriate track of activities. Access-eGov needs to integrate legacy systems which already provide web services or electronic forms.

2.1.3 German pilot

This scenario assumes that a German citizen is currently living in a municipality of Schleswig-Holstein. Her future husband is a Slovak citizen. They have decided that they want to get married within the next four weeks. In case they are able to find a special place for the wedding ceremony (like a ship or a light house) they are willing to wait a little bit longer and also travel for up to 100 km. For example, they want to find out about their options regarding the wedding location, and what kind of legal preparations and documents are necessary. In detail, the questions are:

- What kinds of legal prerequisites do have to exist? (Citizenship, etc.)
- What kinds of documents are needed? (Birth certificate, family records etc.)
- Will the groom, as a foreign citizen, need to supply additional documents?
- Where are these documents available from? (Responsible authority including contact details and office hours)
- Locations available for wedding ceremony, including available dates
- Nearby special locations for wedding ceremony (like a ship or a light house).
- How and where can she book a wedding in one of those locations?
- Any other information that may be of relevance.

The “Responsibility Finder” (an electronic service available in Schleswig-Holstein, providing information about which administration is responsible for a given concern) currently returns the following information:

- Legal requirements in the form of a generally understandable short text.

- Required documents with a short description of what it is and where it can be obtained, including a link to more detailed information like expected time involved in obtaining it etc.
- Which civil registry offices (“Standesamt”) perform weddings?
- A list of other available locations for the wedding ceremony, including a link to broaden the search. This scenario assumes that there is already an operating, state-wide responsibility finder for the territory of Schleswig-Holstein which was built using Access-eGov components and other related technology.

Access-eGov enables information to be shared and integrated from different sources. E.g., in the given scenario the information comes from the following sources:

- The description of legal requirements could come from a catalogue of descriptions provided by the state of Schleswig-Holstein.
- The information about where the required documents can be obtained (addresses, contact details, etc.) could come from each of the administrations that are responsible for each document.
- The information about other available locations can come from administrations as well as private parties offering this kind of service.

For the users this means that they do not have to search for information and services in different places, but at one place only: they visit a single responsibility finder on-line and get all necessary information from apparently one source.

2.2 Security requirements

Security is already mentioned in ISO-9126, but only as sub-item of “Functionality”. In the Access-eGov context on the other hand, security plays a central role. Especially the following security issues are addressed by the Access-eGov platform:

Requirement	Priority[1]	Relevance[2]	Remarks	Consequences of success / failure
Privacy: The system has to keep personal information secure from unauthorized subjects, while at the same time supporting a certain kind of “privacy configurability”, called “personal privacy preferences”: The user is able to create rules, controlling access to and the sharing of his personal information	E	++	• Major task in the Access-eGov security framework	• Failure to meet this requirement means partial project failure
Communication security: All data transmissions have to be encrypted	E	--	• No direct influence on the architecture • Easily achieved through use of standard technology like SSL	
Access Control: Every entry point to the platform is empowered to reject certain connection attempts	E	+	• Careful consideration of possible entry points	• System security could be compromised
Trust between nodes: Every node needs to have a secure and reliable method to assess whether the node it is talking to is a valid one.	E	++	• Major task in the Access-eGov security framework	• System security could be compromised

[1] Priority: essential (E), desirable (D), optional (O)
[2] Relevance: from no relevance (--) to most significant (++)

2.3 Functional requirements

2.3.1 General requirements

According to ISO 9126, the following requirements for software systems arise:

Requirement	Priority[3]	Relevance[4]	Remarks	Consequences of success / failure
Functionality o Suitability o Accuracy o Interoperability o Compliance	++	o	• One of the core requirements for every software product • Success measure for the whole project • No increased relevance to the overall architecture, as each and every component has to fulfil this requirement	• Since this requirement contains suitability and accuracy, failure to meet it means failure of the whole project
Reliability o Maturity o Recoverability o Fault Tolerance	++	++	• Architecture has to provide fault tolerance mechanisms: • By following the SoA paradigm, the components are not tightly coupled, leading to a more robust and fault tolerant system. • The envisioned use of P2P technologies is another building block.	• Failure to meet this requirement leaves the users with a possibly unusable platform or at least a platform which isn't perceived as reliable. Acceptance, especially in the field of governmental services, will suffer.
Usability o Learnability	++	--	• While very important for the project, this requirement has no relevancy towards the architecture as the users will not interact with	

[3] Priority: from the lowest priority (--) to the highest priority (++)
[4] Relevance: from no relevance (--) to most significant (++)

o Understandability o Operability			the platform itself but rather with their personal assistant	
Efficiency o Time Behaviour o Resource Behaviour	+	++	• In the context of the architecture, these requirements mainly address the persistence and discovery/orchestration components • Many discovery operations will be executed, therefore sufficient caching mechanisms and an effective storage layer are needed.	• Users want systems with small response times • Perception of slow systems suffers unproportionally
Maintainability o Stability o Analysability o Changeability o Testability	++	--	• No direct relevance to architecture • Good software engineering practice • Standard quality assurance measures in software development takes care of this requirement	• Failure to address this requirement results in a bad codebase
Portability o Installability o Conformance o Replaceability o Adaptability	o	--	• No relevance to the architecture • No strict relevance to the whole project • Problem for proprietary software products	

2.3.2 User requirements

Not all the requirements that could be collected in the User requirements Analysis [D2.2] have impact on the design of the overall architecture. Some of them are mainly concerned with the presentation layer. The following table lists those requirements, which might be connected to the design of the architecture, together with an assessment of their priority, their relevance to the architecture and comments on the implementation and consequences of success or failure to fulfil those requirements.

Requirement	Priority[5]	Relevance[6]	Remarks	Consequences of success / failure
1.2.1 Access-eGov Personal Assistant adheres to WAI specification.	E	o	• Not strictly an architectural issue, recommendation is to use only tools and software that support the WAI specifications, or at least are no obstacle to their use.	--
1.2.2 Accessibility according to German law ("BITV"), which is based on the WAI criteria.	E	o	• Not strictly an architectural issue, recommendation is to use only tools and software that support the WAI specifications, or at least are no obstacle to their use.	--
1.3.1 Access-eGov is accessible from an Internet kiosk. (Will be installed by user partner.)	D	++	• A web-based personal assistant has to be capable of fulfilling all tasks. Standalone application (for privacy reasons) is optional.	• Disregard of user requirements
1.3.2 Personal assistant sends email and SMS messages in addition to the main web based communication interface.	D	o	• Well designed communication component must be able to handle this requirement.	• Aggravation of overall usability

[5] Priority: essential (E), desirable (D), and optional (O)
[6] Relevance: from -- (no relevance) to ++ (most significant for the architecture design)

1.4.1 Technical solution regarding to security will be prepared in way that ensures accessibility also from public access points (internet cafes etc)	D	+	c.f. 1.3.1	c.f. 1.3.1
1.4.2 Secure authentication and authorization based on a fully qualified electronic signature is available for transactional operations (e. g. invoking services, filling in electronic forms).	E	++	• A flexible authentication and authorization service will have certain plug-ins for various tasks, PKI will simple be one of them.	• Failure to provide this facility could hinder user acceptance • Disregard of user requirements
1.6.2 External partners may participate in the service processes: XML based interface is available, including exchange of user information if user permits.	E	++	• If the message exchange happens in XML, the platform is open to external partners as soon as relevant documentation (schemes, DTDs, API descriptions, …) are opened up. • External partners will have to be given access permission to the platform interfaces.	• Disregard of user requirements
1.9.1 All Access-eGov components provide reliable, trustworthy, and timely information (service providers are responsible).	E	--	• Mainly a task of the information providers, supported by the platform through the possibilities of time stamping data and controlling replication of data • No architectural issue	
1.9.2 Users are able to identify the degree of each of these criteria (reliable, trustworthy, and timely) for themselves by means provided by Access-eGov.	E	--	• Will be solved in the presentation layer using some icons for certain values of those criteria.	
2.1.1 Multi-lingual support for general introduction of services.	E	+	• Use of an internationalisation framework	• Failure to provide this facility could hinder user

				acceptance
2.2.2 User finds step-by-step description of all business processes related to selected services	E	++	• Core functionality	• Failure to meet this requirement means failure of the whole project
2.2.3. The user (a) enters a “search term” which will be resolved to either a life-event, a set of services or a single service by the system or (b) the user selects a task from a predefined set.	E	++	• Core functionality	• Failure to meet this requirement means failure of the whole project
2.3.1 Access-eGov components search for all relevant services and information, based on the identified task, which the user wants to perform.	E	++	• Core functionality	• Failure to meet this requirement means failure of the whole project
2.4.1 According to information provided by the user Access-eGov will generate a life / business event scenario.	E	++	• Core functionality	• Failure to meet this requirement means failure of the whole project
2.5.1 The Personal Assistant works as a personalised system, i.e. it should analyse the information collected from the particular user in order to provide personalised information: – User is asked to provide all necessary information about his/her life event to the system, which will be able to create scenario (iterations are possible). – User is asked also by the system	E	++	• Core functionality	• Without fulfilling this requirement, the system might still be usable via the external XML API • Failure to meet this requirement still means partial project failure

to submit necessary documents or application forms throughout whole process. – Assistant guides the user through the sequence of steps to be taken, offers a personalized "workflow" like a dynamic to-do list for the user – legal basis for this step should be indicated				
2.6.1 The Personal Assistant supports triggering and connecting public service as far as possible: it fills in (personalized) electronic forms, invokes web services, and relates to security issues in context of need for authentication.	E	++	• Core functionality	• Failure to meet this requirement means failure of the whole project
2.6.2 Virtual assistant provides the user with information about the current state of the particular instances of service processes.	E	++	• Platform needs to provide means to store session-based state data.	• Platform cannot support core functionalities
2.7.1 Users may register with the system and / or system requires authentication.	E	+	• Core functionality • Part of the security subsystem	• Failure to meet this requirement means failure of the whole project
3.4.1 Access-eGov provides a way to protect annotated content so that the content provider can restrict who is able to use the annotated content and who has only access to the content (but not the annotation).	O	o	• Wrapper has to take care of this. On systems not using a wrapper, this is the responsibility of the user partner.	

2.4 Data requirements

The properties of the semantic part of the architectural solution will extend the above mentioned requirements:

Requirement	Priority	Relevance	Remarks	Consequences of success / failure
Changeability (Maintainability): The fact that the domain knowledge, which we need to store, changes over time, demands that the expectedly most-frequent changes need to have implications for as little components as possible.	E	++	• Careful consideration of the ontologies involved	• Reliability is at stake
Reusability: Ideally, the domain knowledge storage is totally decoupled from any other components	E	++	• Best design practice	• Maintainability could suffer
Functionality / Suitability: The goal of our architecture has to be, that the user is supported as good as possible in completing his task. For that to be possible, we need to have a great number of concepts, that can be accessed by the platform	E	++	• Careful consideration of the ontologies involved	• Reliability is at stake

3 Conceptual architecture

3.1 Security view

Access-eGov will enable citizens and businesses to access government services via secure channels. In order to facilitate access and to shorten interaction times between several public service agencies, the Access-eGov system can act as a trusted domain to all participating parties. This will empower users to take advantage of Single-Sign-On (SSO) functionality.

Once authenticated by the Access-eGov platform, a user may also be authenticated to enquire services published by other public agencies in the AeG system. Nonetheless, this does not necessarily entail automated access to all other services, because each single public agency will still be empowered to check the user's access rights anew and reject a Single-Sign-On token issued by the platform.

Access-eGov does by no means impose an own Identity Management infrastructure, but allows public agencies to keep their own preferred technology and ways of authenticating users. Further information will be provided in detail in the upcoming documents on security features and on component functionality in Access-eGov.

3.2 Functional view

Although the user requirements are very diverse, the basic functionality of the Access-eGov platform is manageable. We need to annotate services and store them in efficient ways. Those services have to be retrieved according to certain citizen requests and the administrators in public authorities ought to have a possibility to string such annotated services together to form new "meta services", our so called scenarios. To get a deeper insight into the functional aspects of the architecture, two different views are given in the following sections. The Information provider view will look at the functionality from the point of view of an administrator in a public institution, while the Information consumer view will provide a citizen's perspective.

3.2.1 Information provider view

An information provider has three main tasks as described in Figure 2, namely registering new services, annotating services and building generic workflows out of already defined services:

- **Annotate service**
  The public administration official, who in any, except the most trivial, case will also have to be a domain expert, chooses from a set of available ontologies. He uses the therein contained concepts and relationships to mark up the important aspects of the service or website he is currently going to describe. The annotating person will only have limited choice of which ontology to use at a given point in time, since this choice is usually imposed by other, for example institutional, constraints.

Apart from this semantic annotation, the annotator also has to describe the security surroundings of a certain service. This will include information about who is eligible to actually use that service, what form of identification is required up to the point of the privacy policies that the service itself can offer.

Finally, the annotator has to add some general descriptions, to facilitate easier administration of the platform. These properties could include contact details for persons that are responsible for the service's operations and could be part of the non-functional properties.

- **Register service**

  After successful annotation of the service, all generated data is stored; the service can now be retrieved from all participating system components

- **Create workflow**

  A far more challenging task for the public administration official is to build generic workflows based on already annotated services or already existing workflows. First, the annotator has to either find suitable services or already existing, suitable workflows. This is done either through searching or browsing the service and workflow repositories.

  After having found suitable services and/or workflows, the annotator has to order them in a reasonable succession that is imposed by internal procedural guidelines and/or legal conventions.

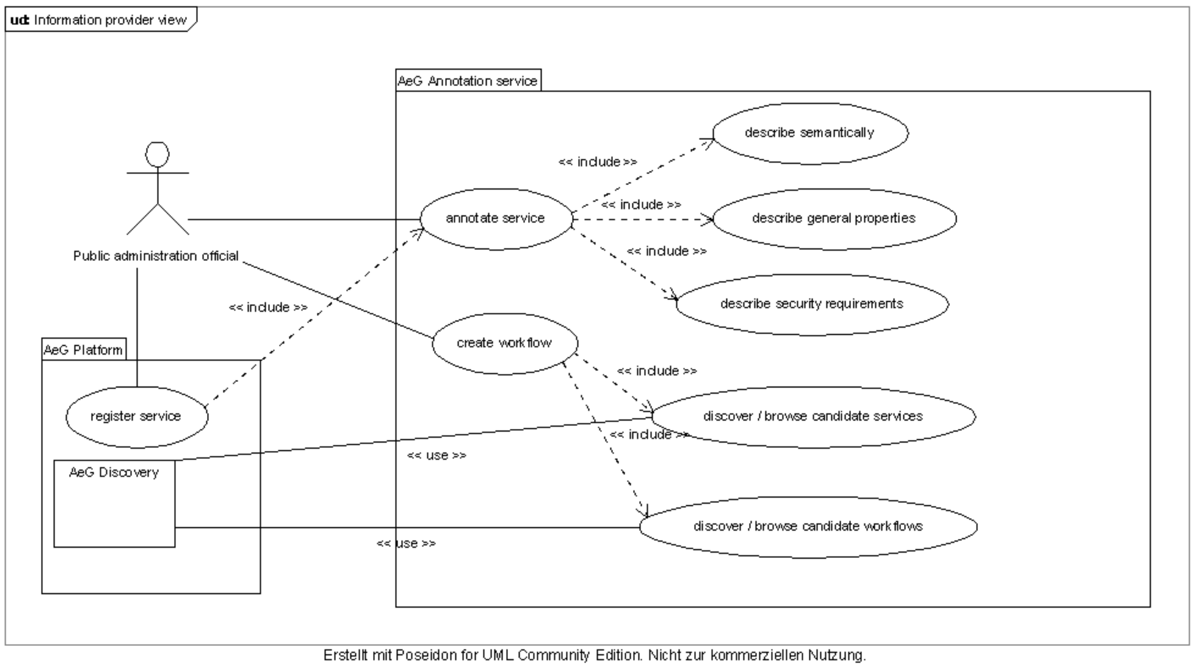
ud: Information provider view
AeG Annotation service
Public administration official
annotate service
create workflow
describe semantically
describe general properties
describe security requirements
discover / browse candidate services
discover / browse candidate workflows
<< include >>
<< use >>
AeG Platform
register service
AeG Discovery
Erstellt mit Poseidon for UML Community Edition. Nicht zur kommerziellen Nutzung.

Figure 2 Information provider view

3.2.2 Information consumer view

Naturally the information consumer (i.e. the citizen or a user of similar status) has a significantly different view on the platform as Figure 3 shows. A citizen has two main possibilities of interaction with the platform; he has to specify his goal and he has to request the platform to execute the retrieved services. Communication with the platform always has to occur through the personal assistant (or an equivalent user client interface), which constitutes from a personal assistant client (e.g. a web front-end) and software parts hidden inside the platform and invisible to the user.

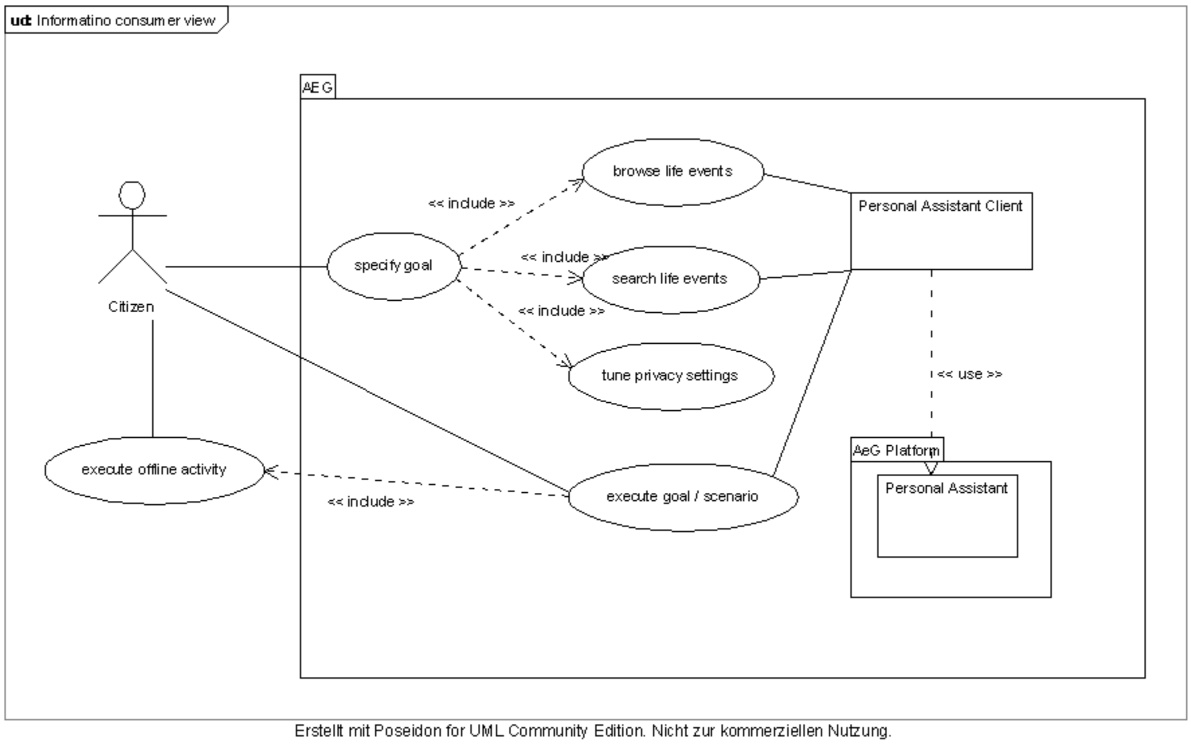

Figure 3 Information consumer view

- **Specify goal**

  The citizen has to articulate wishes to the system and has to tell the actual task that he wants to accomplish. Therefore the personal assistant presents a list of life events or and/or services for browsing and the possibility to search for life events. Once the citizen has chosen one of the offered life events, the personal assistant will translate the selection into a goal.

  It has to be noticed that the life event could either be connected to an atomic service, or to a predefined, generic workflow.

- **Execute goal / scenario**.

  When the user wants to achieve his/her goal, s/he lets the personal assistant start the execution of the retrieved service or workflow. The current progress of this run is always visible to the user through the personal assistant client.

- **Execute offline activity**

  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 notified of the completion of a specific offline activity.

- **Tune privacy settings**

  In the Access-eGov security model, privacy is very important. Therefore the user can, on a very detailed level, decide what user details he wants to relay to whom and under which circumstances he wishes to do so. Since this is configurable for every goal, the user has the possibility to alter privacy settings in the current process, possibly overwriting his default settings.

3.3 Data view

Ontologies are utilized to link real-world concepts (e.g. objects, procedures) and express them in a semantic way that is defined and agreed upon by communities of users (e.g. collaborating organisations). "Ontology" in technical terms constitutes a formal explicit specification of a shared conceptualization. This definition entails a number of essential components which are common for all sorts of ontologies. Ontologies define an agreed common terminology by providing concepts and relationships between these concepts. In order to capture semantic properties of relations and concepts, ontology also provides a set of axioms (i.e. logical expressions in some structured language). We will use three basic ontologies in several parts of the Access-eGov system. These are:

- Life events ontology
- Service profiles ontology
- Access-eGov Domain ontology

Below all the proposed ontologies are described and Figure 4 shows their categorizations and mutual relationships. The upcoming chapters will nonetheless unveil more about the technical specifications of the ontologies used (e.g. the technologies and standards used).

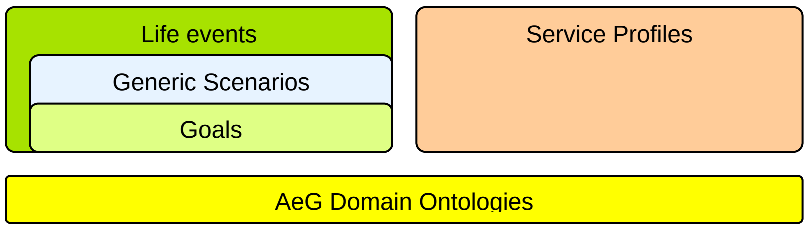

Figure 4 Conceptual data view in Access-eGov

AeG Domain Ontologies are considered lower level ontologies within the overall system. They are used to describe all the relevant domain information related to user's scenarios. That means they will describe functional and non-functional properties of a particular service. We propose web based ontologies that are not necessarily relevant to the web services. When choosing the other two ontologies mentioned in the figure above, it would be favourable to describe them in a standardized way so that all system partners will be able to process them. A more detailed description will be given when relations between ontologies are available in upcoming documents that are specifically relating to this issue.

The other proposed two ontologies in Access-eGov system will be the “Life events” and “Service Profiles” ontologies and are used to describe more abstract data. They are not simple web ontologies, but extended with semantic descriptions of possible life events (Life Events ontology) or (web) services (Services Profiles ontology). A process model is used:

a) to guide citizens or businesses to achieve specific goals, and
b) to co-ordinate activities that are performed by all actors - citizens, traditional public administration services and web services.

Life events denote specific situations (in the life of a citizen or a life cycle of an organization) that require a set of public services to be performed (cf. chapter 4.4.1 in this document). It contains information about particular “Goals”, which can be grouped into several “Generic scenarios”.

In general, these three ontologies describe several aspects and levels of the same real world data. All of them denote services (web or traditional) and the way how they are used. Thus we propose to use these three ontologies in order to describe all relevant data inside the Access-eGov system. The domain ontology will be used to describe the lower (i.e. technical) level of the Access-eGov system, whereas the other ontologies will be utilized to denote more abstract system levels just as service description. On the upper, most abstract level, sequences of goals $\rightarrow$ *scenarios* $\rightarrow$ *events* can be described. In order to allow interaction and deduction between the different data models that underlie these ontologies, several layers of mediation will need to be introduced in Access-eGov.

4 Logical architecture

4.1 Possible architectural design choices

This chapter gives an overview of different application architectures and middleware component technologies used for application communication over networks.

The architectural middleware environments that are most visible today are CORBA, Enterprise JavaBeans, message-oriented middleware, XML/SOAP, COM+ and .NET. However, since late 1990s, the middleware landscape has continually shifted. Today the most dominating architectural approaches to be realized are the classical Client/Server and the relatively new Peer-to-Peer-approach. The most common middleware philosophies for use in distributed software architectures are Web services (used in conjunction with Service-oriented architectures) and so-called Advanced Distributed Components (as a meta-category for Enterprise JavaBeans, COM+, .NET and the like).

In this chapter we will evaluate the adequacy of possible architectural approaches for the Access-eGov platform. The adequacy is being measured based on the architectural requirements as set out in the Technical Annex and in the user-requirements. Further information about these architectural approaches can be found in the Annex to this report.

4.1.1 Client-server architecture

The three tier architecture is used when an effective distributed client/server design is needed that provides (when compared to the two tier) increased performance, reusability and scalability, while hiding the complexity of distributed processing to the user. These characteristics have made three layer architectures a popular choice for Internet applications and net-centric information systems.

Although three tier architectures emerged to overcome the limitations of the two tier architecture (such as limited scalability and maintenance), both technologies lack the openness as required from modern information systems operating in heterogeneous environments. The three tiers are tightly coupled, making it difficult for external third parties to gain access to services offered.

Also, distributed systems based on a pure client/server paradigm proved difficult to maintain, as updates of the client software will have to be disseminated across the network.

	Strengths	**Weaknesses**
Present	• Proven and grown-up technology • Wide support in IT community	• Complex implementation • Out-dated • Single-point-of-failure • Resource demanding
	Opportunities	**Threats**
Future	• None	• Limited scalability

		• Congestion prone • Enormous effort to update every single client software package in an existing network

Considering static client/server architecture for Access-eGov is not advisable due to the enormous effort in adding new nodes to a dynamically growing network offering electronic services. Therefore, we will need to consider a technology allowing components to be more loosely coupled in order to guarantee easy scalability on a large scaled network.

4.1.2 Advanced distributed components

Regarding the development of a large distributed system based on EJB components, the initial hurdle is relatively high: legacy systems may only be integrated with a considerable amount of customization effort. Each public agency (or data centre that is responsible for hosting public agency content) will then need to run EJB containers with dedicated applications handling interactions with each single service.

	Strengths	**Weaknesses**
Present	• "Write once, run anywhere" portability • Mature, proven de-facto industry standard • Runtime environment available for almost all platforms • Some container implementations are available as open source	• Proprietary technology and only partially standardized • Complex and difficult to maintain • Hard to migrate solutions between different container implementations
	Opportunities	**Threats**
Future	• Strong communities guarantee ongoing development	• Future licensing policy unsure

Even if the EJB specification does not require the component provider ("Bean Provider") to have any programming knowledge for concurrency, transactions, and other services, s/he must first accomplish a detailed analysis of all the enterprise beans' methods before starting the configuration. Changing the values defined by the Bean Provider for these attributes is highly error-prone.

Though beans in EJB architecture are independent from the actual server implementation and may be migrated from one server to another without major changes, their design and programming philosophy is to a certain degree dictated by the architectural paradigms as set out by Sun Microsystems.

Considerable effort would need to be undertaken in order to enable legacy systems in Access-eGov to communicate using these highly complex technologies. Using advanced distributed component technologies as described above will imply a large amount of work to be spent on tailoring legacy systems to interact with these newer technologies.

4.1.3 Service oriented architecture

The ease with which Web services can be implemented and the ability to access them from any platform, local or remote, has led to their rapid adoption by system management bodies as virtualization agents that provide common manageability interfaces to disparate resources. For example, a Web service might be designed to “represent” a particular device or a legacy application, accepting control and monitoring requests through standardized interfaces, communicating with the resource through its native interface, and returning the result to the requester in a standard XML type or user-specific format using XSLT technology.

This degree of openness makes the service oriented architecture approach best suitable for use in Access-eGov. Wrapper applications will then need to be installed on the side of the participating public administrations. The details of such a wrapper technique will be subject of the deliverable to be produced within Task 3.3 (Access-eGov components functional description).

	Strengths	**Weaknesses**
Present	• Use of existing Web technologies • Relatively small amount of code needed to expose existing legacy application • Fast amortization of costs once it is up and running	• Difficult to implement in highly decentralized environments • Higher initial development costs than for other distributed architectures
	Opportunities	**Threats**
Future	• Service re-usability • Highly interoperable • High scalability of the network • Best solution in case of possible process transformations	• Use of SOA may result in a multitude of different interface formats (see Danish OIOXML)

Building Access-eGov using Web services as basic technology will most likely reduce platform complexity, since application complexity will be divided more or less equally between the service providers (dealing with attaching legacy systems via XML-interfaces) and the platform that’s dealing with service registration, discovery and delegation of service requests. In comparison to using advanced distributed component technology, there will be less effort to enable legacy systems to interact with the overall system and other services.

4.1.4 Peer-to-peer architecture

The peer-to-peer architecture offers the promise of harnessing the resources of vast numbers of Internet hosts. The primary challenge facing this P2P architecture is efficiently locating information distributed across these hosts in a decentralized way. The reason for P2P’s popularity in file sharing – fault tolerance, scalability, and ease of deployment – is its good model for distributed data management.

Compared to the classical client/server architecture, P2P is easier to maintain and deploy, as only one type of software needs to be updated and installed.

In a public service environment, infrastructure stress would be reduced to a minimum, since virtual overlay network will coincide with physical Internet topology of the nodes. Consequently, this will render the responsiveness of the overall system extremely efficient.

	Strengths	**Weaknesses**
Present	• No specialized equipment needed • Inherently scalable network • Self-organizing network • Network redundancy	• “Unreliable” infrastructure • Lacks of in-built security features • Expensive inter-domain traffic to keep network structure consistent
	Opportunities	**Threats**
Future	• Inexpensive infrastructure • Easy integration of new services/nodes • Low entry barriers • Easy maintenance and updating	• Lack of reliable security mechanisms • Stability/QoS

For Access-eGov, it might be useful to consider service-offering public service resources as peer-nodes according to the above-mentioned peer-to-peer paradigm. This will allow easy, automatic and un-attended joining of new resources/nodes to the network once it is in operation.

4.1.5 Architecture conclusion

All the architectures outlined above impose several obstructions to the future overall system. Especially the classical client/server approach and the highly specific advanced distributed components technologies are likely to limit the system architect’s freedom by proposing a tight coupling of components and the use of certain technologies.

Implementing Web Services in conjunction with service oriented Peer-to-Peer architecture seems to be the most promising approach with regards to interoperability between different types of legacy systems used within public service backend. The implementation of Access-eGov based on these technologies seems to be far more feasible, but nonetheless challenging, since further research in the field of distributed application architecture is still necessary.

Since Access-eGov is envisioned to build a service oriented platform, it is supposed to be highly modularized and shall be logically composed of a number of components interacting with each other (confer to structural view for more details on service interaction). Figure 5 gives an overview of the system components.

The overall Access-eGov system (also called the platform) may be sub-divided into three major component groups:

- the AeG Infrastructure itself,
- the AeG Personal Assistant client and equivalent end user interfaces,
- AeG Administration and Management Tools (e.g. Annotation services) which are not integral parts of, but affiliated to the AeG Infrastructure.

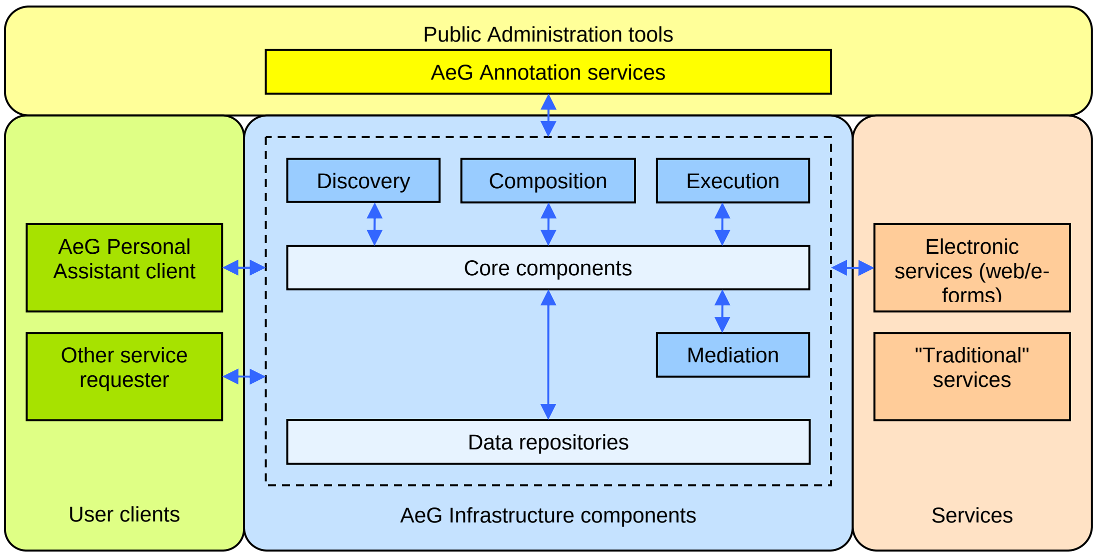

Figure 5 System architecture of the Access-eGov system

The actual services are still hosted under responsibility and on the premises of the participating public agencies or their respective data centres. They are simply made available through AeG and thus do not form an integral part of the overall AeG system. Those are either electronically available (directly via web service interfaces or web forms) or represent "traditional" office services that may merely be described and registered within AeG. Only executable services will dispose of an electronic XML-interface to the AeG 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 they will find available as web-based application.

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

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

Public agencies may choose which of the above stated AeG Infrastructure components they wish to install on their premises or data centres. Such a "local" installation of AeG 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 bring to the overall AeG system.

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

4.2 Security view

Security mechanisms in Access-eGov are split between several system components, notably between the Personal Assistant and the platform. The AeG Personal Assistant will handle authentication credentials provided by the user and forward them to the respective public service agency. After choosing a strategy and before allowing access to the services, the Personal Assistant lets the platform check the user credentials.

Therefore, the Security component (one of the dynamic components of the platform) takes into consideration the security level as required by the public service provider (depending on the security scheme qualifiers as can be found in the Security Repository) and the validity of the credentials provided. Only then, a user's Personal Assistant may be allowed to enter the platform and forward the service requests to the public agency in question.

Security checks will thus be handled by the platform itself, with the Personal Assistant acting as thin client front-end with only limited decision and calculation functionality. Once a user's Personal Assistant has been authenticated, the platform issues a security token that the PA can use for Single-Sign-On purposes where public agencies accept them.

The implementation details and further information will be provided in detail in the upcoming documents on security features and on component functionality in Access-eGov.

4.3 Functional view

Discovery Use Cases

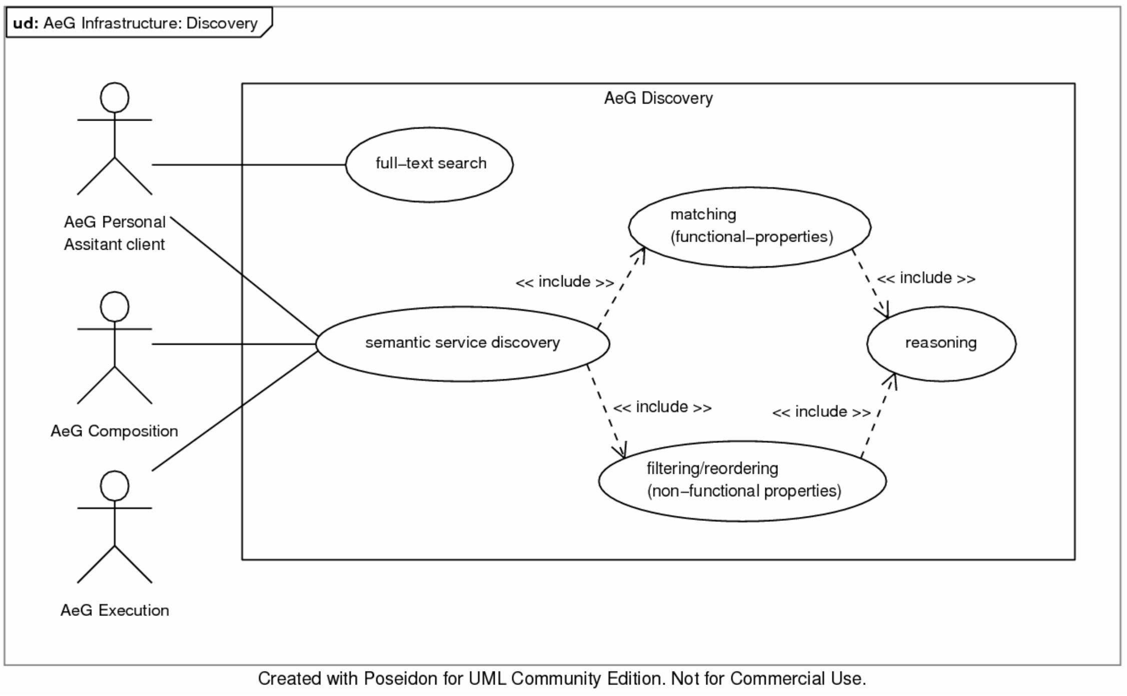
ud: AeG Infrastructure: Discovery
AeG Discovery
AeG Personal Assitant client
AeG Composition
AeG Execution
full–text search
matching (functional–properties)
semantic service discovery
reasoning
filtering/reordering (non–functional properties)
<< include >>
<< include >>
<< include >>
<< include >>
Created with Poseidon for UML Community Edition. Not for Commercial Use.

Figure 6 Use case "Service Discovery"

Use case	Description
Full-text search	First, the user query (e.g. life event description) is matched (using some engine) against basic textual descriptions of pre-defined goals in PA client. An index structure created from these textual descriptions is used for matching purposes. Usually (see orchestration scenario), a set of ranked pre-defined goals is found.
Semantic service discovery	Assuming that a predefined goal was selected, the discovery process is performed by the Discovery component. It uses domain ontology of functional service description as well as life event/goal ontology to find services which can fulfil the goal.
Matching (functional-properties)	Initial task in the semantic service discovery involves semantic matching between the functional-properties of selected goal and the functional-properties of services. A set of services can be obtained.
Filtering/reordering (non-functional properties)	If the set of discovered services is large, filtering/reordering should be used. The use of the discovered services can/should lead to fulfilment of the goal, but there can be many of them, which are actually not worth from a user's point of view (e.g. the location problem). Preferences obtained from the user profile as well as some other preferences obtained from additional questions might be used to filter results as well as for reordering the list of the services according to the rank derived from the mentioned preferences.
Reasoning	The discovery component will invoke the reasoning component, which will use domain knowledge described in the domain ontology. This component can infer new knowledge (i.e. relevant concepts which help in the process of semantic matching).

Composition Use Cases

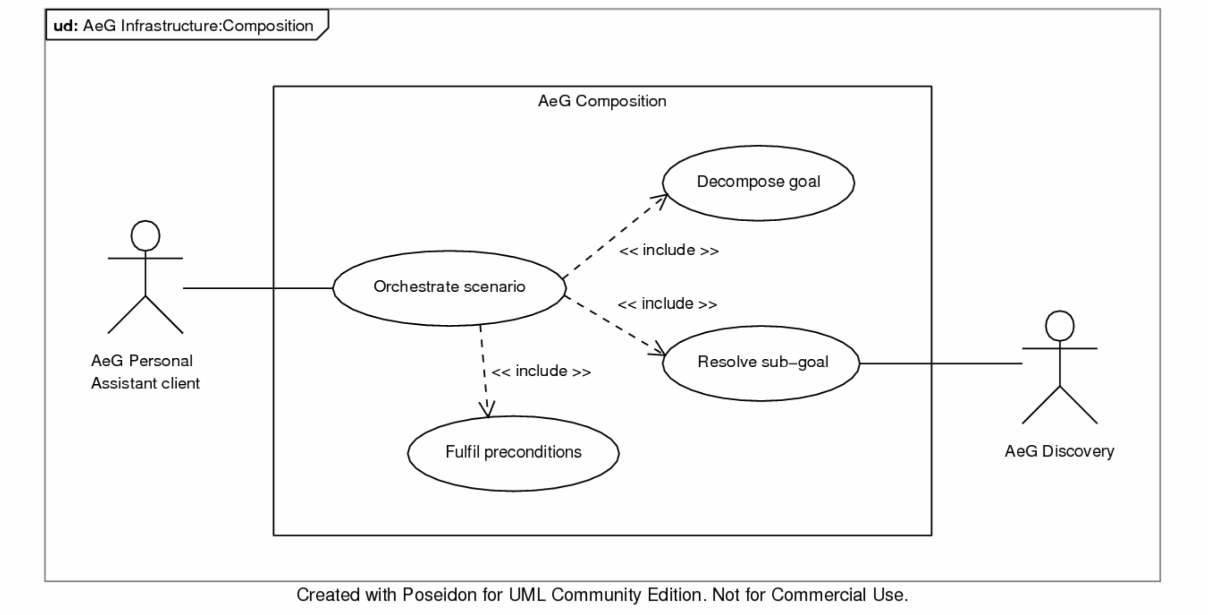

Figure 7 Use case "Service Composition"

Use case	Description
Orchestrate scenario	In case the AeG Discovery component cannot find atomic services, which are able to achieve a requested goal, the goal description is delegated to the AeG Composition component, which will try to orchestrate existing services to the new scenario to achieve the goal.
Decompose goal	If the goal is complex and has multiple effects, AeG Composition will decompose this goal to sub-goals, which can be achieved by atomic services. The current specification of the AeG Composition component includes a semi-automatic approach based on the generic scenarios defined for the life event categories.
Resolve sub-goal	Sub-goals are represented in the process model of the generic scenario as abstract activities, which have to be resolved before the scenario is executed. To resolve sub-goals, AeG Composition invokes the AeG Discovery component.
Fulfil preconditions	In case not all the inputs or preconditions of the resolved services are met from the outset, automatic "chaining" of services will be used to fulfil preconditions.

Execution Use Cases

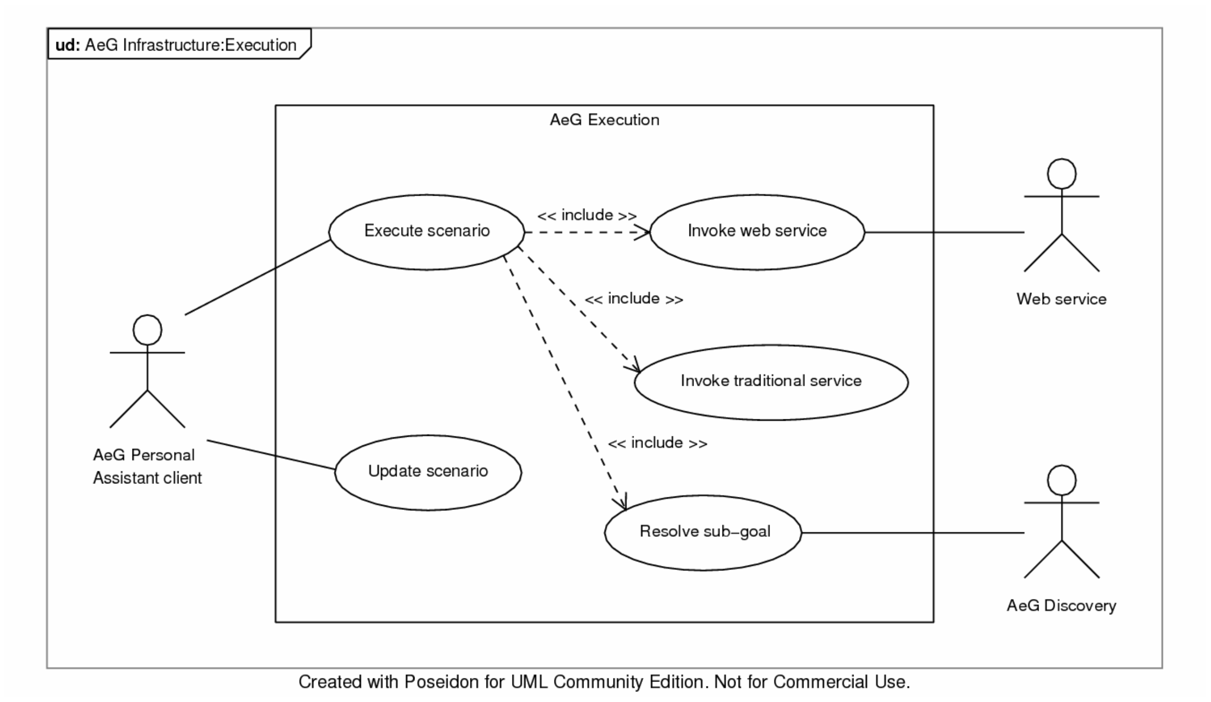
ud: AeG Infrastructure:Execution
AeG Execution
Execute scenario
<< include >>
Invoke web service
Web service
<< include >>
Invoke traditional service
AeG Personal Assistant client
Update scenario
<< include >>
Resolve sub-goal
AeG Discovery
Created with Poseidon for UML Community Edition. Not for Commercial Use.

Figure 8 Use case "Service Execution"

Use case	Description
Execute scenario	AeG Personal Assistant client sends initial data for the required inputs. Execution component creates and executes process instance for the user scenario according to the specified process model.
Invoke web service	If some inputs are missing, Execution component will suspend the

	scenario process and send the semantic description of the required inputs to the AeG Personal Assistant client. The AeG Personal Assistant client will resolve the missing inputs from the user profile, or generate web forms and invite the user to enter the input data and update the state of the process execution.
Invoke traditional service	Execution component will suspend the execution and send to the AeG Personal Assistant client a semantic description of the service (with all information about the service provider and required inputs). The citizen can use the AeG Personal Assistant to update and restore process execution.
Resolve sub-goal	If the selection of a concrete service instance for the sub-goal specified in the scenario model is depending on outputs of previous process activities, it is not possible to resolve this sub-goal in the phase of the scenario orchestration. In this case, selection has to be deferred at execution time and the AeG Execution component will invoke AeG Discovery to find a service required to achieve this sub-goal.
Update scenario	Citizen will use the AeG Personal assistant web interface to provide information about the state or the result of some "traditional" service. After updating context data, the Execution component may restore execution of the scenario process and continue with the subsequent activities.

4.4 Data view

Structural correlations of data held (persistently) and used within the Access-eGov system can be seen in Figure 9 illustrated below. The following chapter describes the relations between the several data entities in order to provide a more detailed view on the platform data structures. All the presented concepts are preliminary and will be specified in more details in the upcoming reports. It is noteworthy that definitions of data items are necessary for understanding of dependencies between AeG platform components.

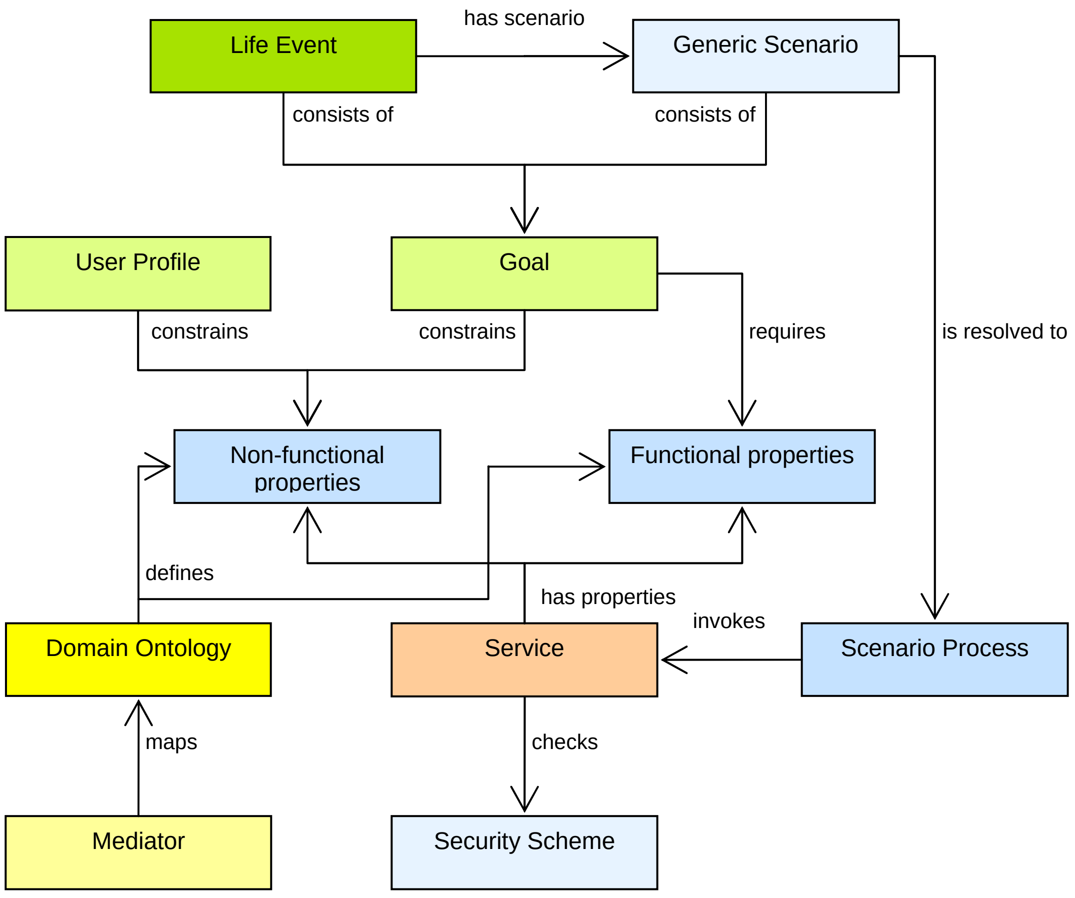

Figure 9 Logical data view on Access-eGov

4.4.1 Life events

Life event (LE) denotes a specific situation (i.e. event) in the life of citizen or a life cycle of organization that requires a series of public services to be performed.

Life events can be categorized into groups and may be organized in multiple hierarchies. Using the AeG Personal Assistant portal site, user may "browse" or navigate through life event categories in order to select an appropriate life event.

A life event may be assigned multiple goals, which will formalize user needs. Life event's goals can have specified optional preconditions, which allow users to customize their specific life event. Preconditions are specified as logical expressions with input variables provided either explicitly by the user or from the user profile (preconditions are not dependent on service invocation). More complex life events can organize goals with more complex workflow models specified as generic scenarios (see the following sections).

4.4.2 Goals

Goal specifies those objectives that client might have when consulting a service, including functionalities that the service should provide from the user's perspective. Goals formalize user needs by specifying the requested outputs and effects. This is declared in the same way as service functional properties.

The reader may notice that there is no relation between user's goal and service's capability specified with functional properties. Goals are only logically matched against service's capability, but the data structures are completely independent from one another. They are not directly connected to service- and user-related data.

4.4.3 Service profiles

Service profile specifies what the service does provide from user's perspective and is used by the public administration to advertise services. Service profile consists of non-functional and functional properties.

Functional properties describe inputs, outputs, preconditions and effects of the service (IOPEs). They are specified as logical expressions, which consist of terms constraining type and property values of various resources required for or provided by the services. Types used to specify functional properties are defined in the domain specific resource ontologies.

Non-functional properties describe semi-structured information intended for requesters 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, quality-of-service, security, trust, etc.). Structured non-functional properties are specified by domain specific ontologies (i.e. organization structure of the service provider) and general (e-government independent) ontologies (i.e. types used to describe quality-of-service, security or trust).

In the process of the service discovery, functional properties of goals and services are semantically matched by the AeG 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 a requester's preferences.

4.4.4 Scenario process model

Process model is used:

a) to guide citizens to achieve specific goals, and
b) to coordinate activities performed by all actors - citizens, traditional public administration services and web services.

The process model is represented as a collection of activities designed to produce a specific output for a particular requester, based on a specific input. In that sense, activity is a function or a task that is carried out over time and has recognizable results. The process model of the orchestrated scenarios should provide support for the following types of activities:

- *Atomic activities* are directly executable (by feeding them with the appropriate messages). Atomic (also called "*non-composite*") activities have no sub-processes and are executed in a single step from the perspective of the requester. Each atomic activity has assigned a concrete service (service instance). In the case of web services, each service instance must be assigned a grounding that enables AeG Execution component to construct messages and invoke this service (see AeG Execution component description in Section 4).

- *Composed activities* (also called "*composite*") are decomposable into other (non-composite or composite) activities. Their composition can be specified using logical control constructs such as sequences, if-then-else branching and iterations; this will allow for complex operations and execution looping in a workflow scenario.

- *Abstract activities* are not invocable and are not associated with a concrete traditional or web service. Abstract activities are used to specify functional and non-functional properties required for services to solve sub-goals specified in generic scenarios. Abstract activities are resolved to invocable activities in the planning phase (see AeG Composition description) or during the execution phase (see AeG Execution description). Resolving of abstract activities means that some abstract activity is assigned to a concrete service instance; or an abstract activity is replaced by the activity composed only from invocable (non-abstract) activities.

- *Activities assigned to human actors* (e.g. citizen) are associated with traditional services or can be used to model citizen activities which are related to a specific life event but are out of the scope of the public administration services.

Besides supporting the described activity types, the following facilities were identified as useful for a process model to provide support for modelling orchestrated scenarios:

- compatibility with the standard process modelling notation (i.e. BPMN) in order to visualize scenarios to the users and to use standard tools for modelling,
- compatibility with the proposed standard workflow modelling languages (i.e. WS-BPEL)

4.4.5 AeG domain ontologies

The AeG domain ontologies will be used to represent functional and non-functional properties of a particular service. This section provides a preliminary description of the e-Government specific domain ontologies for AeG.[7]

At this stage it is assumed that the following domain ontologies will be needed to describe the concepts for non-functional properties of services for AeG:

[7] A more detailed analysis will be provided in subsequent Access-eGov deliverables (D7.2 and D7.1).

- *Fees*: Describes fee that citizen has to pay in order to use the service.

- *Forms*: Services may require information and/or they might provide information in the form of documents or forms. The *Forms* ontology will be used to describe these kinds of (both mandatory and optional) input and output of service.

- *Input and output artefacts:* For inputs and outputs that cannot be described with the *Forms* ontology (for example, an artefact like a license plate), AeG will provide a special ontology that can be used to describe these special kinds of inputs and outputs.

- *Administration:* Every service is provided by one or more administrations. At least the following information related to service provision of an administration must be described:

  – *Responsibility*: Two administrations may offer the same service, but only one administration is responsible for providing the service to a particular citizen. The responsibility of administration can be divided into at least three different kinds: *spatial responsibility* describes the geographical area within which an administration is responsible for providing a certain service, *temporal responsibility* describes the time during which the administration is responsible to provide the service (this may not be confused with the office hours, which is the time the service is provided to citizens), *subject-matter responsibility* describes professional area of responsibility of an administration (for example, the civil office in Germany is *professionally responsible* for marriages, but not for income taxes). Depending on legal requirements in effect, which administration is responsible for providing a particular service to a particular citizen can depend on any of the three kinds of responsibility or a combination thereof.
  – *Office hours/availability:* Describes the time when citizens can request the service. This is especially important for the description of traditional (non-electronic services).
  – *Address and contact information:* Describes the address, telephone number and other contact information of the service provider (for electronic services) or the location where the service can be requested (for non-electronic services).
  – *Physical accessibility constraints:* Describes any physical accessibility restrictions (for example, accessible by wheel-chair) that apply to a certain administration (service provision location). Only needed for non-electronic services.

Each domain ontology may have as many mappings as the number of ontologies in the Ontology repository. This way, ontology can have m:n-relations to other entries in Ontology Repositories. All ontologies that the Access-eGov platform has to be familiar with will be stored in persistent repositories that are accessible to all peer-instances. All ontologies will consist of a core set of public service concepts to sufficiently describe services. Actual ontologies that are used for annotation process and for lookup during automated service retrieval will need to be provided by the respective public service provider.

4.5 Structural view

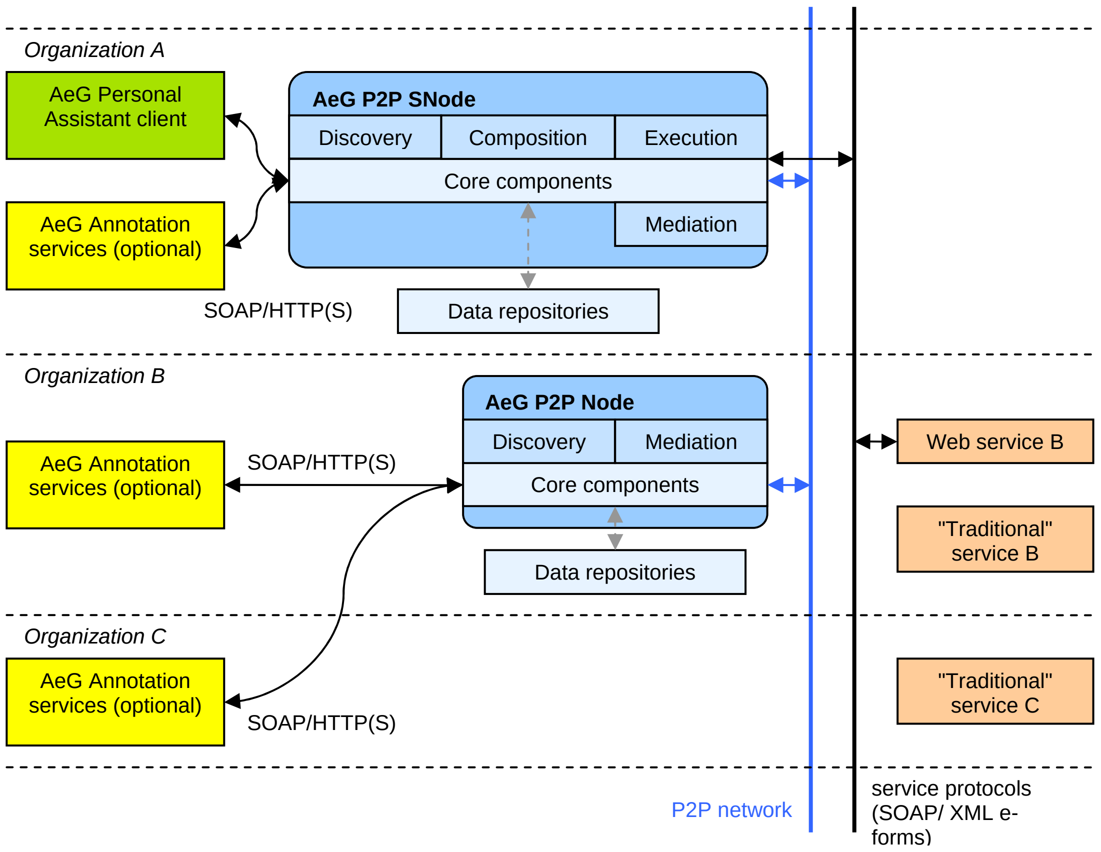

Figure 10 Structural view on Access-eGov

Figure 10 depicts physical division of the system into several levels. On one hand, there is the vertical level division. Three vertical levels are depicted:

- user level (on the left hand side of the scheme),
- system level (in the middle of the scheme), and
- web and traditional services level (on the right hand side of the scheme).

On the other hand, the horizontal levels, divided by the horizontal dashed lines in the scheme above, represent several possible AeG installations on public administration premises to access the AeG peer-to-peer-network. Only three sample Organizations are depicted in Figure 10, but practically there can be several horizontal levels in the real world applications.

Finally four different types of communication are depicted between the several levels and are represented by arrows in different colours. Communication in this chapter means physical data and information interchange.

Vertical levels
Let's explain the vertical levels first. On the left hand side in the scheme above, the user level is situated. Two colours are used on this vertical level - green and yellow – to distinguish between two types of users and user interfaces that are depicted in that level:

- *User citizen* uses the system via the AeG Personal Assistant client (green boxes)

- *Public administration,* the AeG administrators on the Public Administration site, who annotates the services to be used in the AeG Node (see below what AeG node means) using AeG Annotation Services interface.

There is another user type that is not shown in the scheme, namely the administrator of the AeG system. But this user will interact with the system only during the installation phase, not within the lifecycle of the system.

The AeG system level is depicted as blue-coloured middle vertical level of the scheme. The AeG system will be represented by nodes called AeG P2P Nodes. These nodes are connected via peer-to-peer-network. Each AeG Node consists of modules and components as shown in chapter 4.1.5. According to the functionality needed in a particular AeG P2P node, the actual number of installed optional components may vary. More information on the AeG node functionality will be unveiled in the description of the horizontal levels of the scheme below.

On the right hand side of the scheme there are Web and Traditional services. This is the level, where web and traditional services from the provider side are situated.

Horizontal levels
As mentioned before, horizontal levels in the scheme above represent particular organizations. Note, that the term particular organisation does not necessarily mean a different organization. It can also represent a department within one organisation. In such an organization one AeG P2P node will be set up, which can be administered by this organization, but should cooperate over the P2P network with other AeG nodes installed in different organizations. Every AeG node may lack some functionality components according to organizational needs or restrictions. There are three possible installation types depicted in the scheme.

Organization A is an organization with the fully featured AeG node installation. This node consists of all the AeG modules and components. It provides services to both the AeG Personal Assistant and AeG Annotation services user interfaces.

Organization B does not provide a user interface to citizen users, so there is only limited AeG node functionality implemented. Users can use the web or traditional services described in this AeG node via other (fully featured) nodes (using P2P).

Organization C does not have an AeG node installed; it only annotates its web and traditional services using an AeG node outside the organization. Such an organization will only need to be able to access AeG Annotation services.

Finally let’s describe different communication types proposed in the scheme. Black arrows on the left hand side between the user and system level are to represent HTTP(S)/SOAP communication. On the right hand side, blue arrows represent the P2P network used to communicate between the Nodes. A particular AeG Node can act independently of other nodes, but if there is a need of communication between two AeG Nodes (a user connected to one AeG Node needs a web service described at another AeG Node) the network will handle this communication. Black arrows on the right hand side represent communication between particular nodes and the services (web or traditional). There is another type of communication proposed in the scheme, represented by grey arrows. It is namely communication between one AeG node and its Data repositories, as the data repositories will represent data in several formats that do not have to be stored on the same physical device as the AeG Node itself.

4.5.1 Core components

- SWS ontology API - in-memory object model for semantic web services ontologies; the persistence layer will load ontologies from the repository into SWS API objects used by other components (API can be used also in personal assistant and annotation services).

- Connection manager - creates web service entry point which makes infrastructure components available to AeG Personal Assistant and Annotation services

- Security 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

- Notification services - distribute internal messages required to coordinate activities of AeG Infrastructure components and notifications for the 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).

4.5.2 Discovery

In the process of service discovery, functional properties of goals and services are semantically matched with each other by the AeG Discovery module to select services which can 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.

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 with 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 the actual input values are available for the discovery process, matching can be based on rich semantics, which capture 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. In this way, AeG Discovery can achieve highest precision, because it is possible to select only those services which are able to provide the requested outputs considering 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 AeG Execution component can collect as many data as it will be possible for the AeG discovery.

Under the assumption that eventually a large number of services will be available to the AeG Discovery component, a faster filtering of relevant services before evaluating them one by one is essential to make discovery scalable. This can be implemented with indexing the terms in functional properties (IOPEs).

When the AeG Discovery component selects services according to the functional properties, non-functional properties specified by the requester are used to additionally filter or reorder discovered services according to the requester preferences. The fact whether a particular non-functional property will be used as a constraint for filtering or as a criterion for reordering, will depend on the request context and it has to be specified in the discovery request.

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

Besides semantic discovery, the AeG Discovery component will provide an interface for full-text search of services and life events/goals. Full-text queries will be matched with the text extracted from unstructured non-functional properties (i.e. name, description, etc.). Full-text index of the various data objects will be managed in the corresponding data repository.

4.5.3 Composition

In case the AeG Discovery component cannot find an atomic service which can achieve a requested goal, the goal description is delegated to the AeG Composition component, which will try to orchestrate existing services to a new scenario that may solve this goal.

Although there are many initiatives to define industry standard languages for web service orchestration like BPEL, they still have limited capability to support static service composition. The AeG Composition component provides support for a dynamic composition of services which is not based on a static workflow predefined for a respective life event.

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

- *Fulfilling preconditions*: a service that can provide the desired effects and outputs exists, however, not all preconditions of this service 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.

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

A generic scenario will specify the process model which will guide the decomposition of multiple effects specified for a life event to the sub-goals achieved by the atomic services. Sub-goals are then resolved with the AeG Discovery component which will match sub-goal descriptions with existing services (most likely with additional planning inside the AeG Composition component) to fulfil all given 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.

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

Since we assume that complete knowledge is unsuitable for service composition, the planning algorithm must be able to detect such a situation and has to implement procedures, which will acquire the missing knowledge. For example, if the selection of a concrete service for the abstract activity is dependent on the outputs of previous activities, it is not possible to resolve this abstract activity at composition time and selection has to be deferred to execution time.

4.5.4 Execution

The Execution component is responsible for invoking discovered atomic web services or for invoking composed services of orchestrated scenarios. For composed services, the Execution component creates process instances according to the specified process model and executes it with initial input data provided by the service requester.

Created process instance is associated with the execution *context*, which contains values for all process variables (ontology instances) and the current state of the process activities (i.e. which activity is not active, has already started or has already finished). Process variables consist of input from the service requester, shared variables used for the control and data flow, and outputs of the already finished activities. The context container is persistent and used to store the process state when a process was suspended to wait for the asynchronous invocation of a web service or for updating due to human interactions.

Operations in that context can also be used to define a communication protocol between the Execution component and other components which will update or inspect the state of the executed processes (i.e. AeG Personal Assistant client).

If the service invoked in the process activity has specified a delivery time (as a non-functional property), the Execution component will create a timeout-timer for output-delivery. When the expected or the maximum time is exceeded, the Execution component will notify the requester client using core services functionalities.

In case the process activity invokes a web service, the Execution component will use the grounding mechanism specified for the web service to translate inputs and outputs from the abstract semantic layer to form a concrete specification of how to interact with the web service. Service grounding specifies all details about the communication protocol, message formats, transport and service addressing. Although service grounding may support different message specifications, support for the widely accepted WSDL specification is preferred for the AeG platform.

4.5.5 Mediation

The role of the AeG Mediation component is to reconcile the semantic and data heterogeneity problems that can appear during discovery, composition or execution. For the Access-eGov platform, it is not expected that public authorities 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 user request, namely during 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. AeG 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 one life event refers to many goals, each goal can be described with a 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 AeG Discovery component matches the semantic description of a service with the semantic description of a goal, a service-to-goal mediator will align the different goal and service ontologies. 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 reverse mapping from the union ontology which specifies the orchestrated scenario for the life event request to the ontology that is used to describe particular services involved in service orchestration. This mediation is required for invocation because the 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 will take the reverse direction for the outputs.

4.5.6 Persistence repositories

The Persistence layer of the AeG Infrastructure provides interfaces to store and retrieve various data objects used by other components. At this stage, the following repositories were identified in the design of the AeG platform:

- life events/goals repository - manages goals and generic scenarios associated with the life events;
- service repository - stores descriptions of web services, traditional services and composed services registered for specific orchestrated scenarios;
- ontology repository - stores domain ontologies and associated mappings for mediators;
- process context repository - manages the context (i.e. state) of the processes executed for the orchestrated life event scenarios;
- security data repository - stores user login information and access rights.

Besides the previous repositories that are already included in the AeG Infrastructure, the AeG Personal Assistant client will manage a user profile repository and it may additionally manage repositories intended for caching of other data objects like life events and goals in order to improve user interface responses. Changes in the AeG infrastructure repositories are propagated to the AeG Personal Assistant client using the AeG Core components notification interface.

To improve retrieval efficiency, description of the services should be indexed according to the logical terms extracted from the structured service properties. The term index will be managed by the service repository and used by the AeG Discovery component. Service and life event/goal repositories will also manage indices for full-text search. Full-text search of goals and services is based on the keywords extracted from the unstructured non-functional properties. Indexing of keywords is handled transparently by the repositories which will provide query interface used by the AeG Discovery component.

4.5.7 Annotation services

The Annotation service as used within Access-eGov will consist of a web-based application that is not an integral part of the AeG 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. For this purpose, the web application provides crawling capabilities to allow for easy inspection of existing content, which can be annotated in the aftermaths.

The AeG Annotation module will be allowed database access via web service interfaces to the respective Repositories within the Persistence Layer (notably the Ontology and Service Repository) in order to register services and publish their respective descriptions. The creation, modification and editing of these semantic descriptions is controlled by the security subsystem.

This web application will also have modules for management of life events and goals. Information will be stored in separate repositories and later can be accessed by the Personal Assistant client to allow more user friendly and effective navigation within life events and goals. The security subsystem will provide means for fine-grained access control and also for modifying or deleting any existing, or creating new life events and goals.

With the help of the AeG Discovery component, the annotation service can also retrieve services with properties matching the required properties. Those services can then be chained together by the user to create workflow scenarios. These workflows can be rather generic; the downstream orchestration component will be able to customize these workflows to a certain degree to fit instances of any specific life-event or goal.

4.5.8 Personal assistant client

AeG Personal Assistant (PA) client is envisioned to be a thin client with web user interface that will in conjunction with certain AeG Infrastructure components provide functionalities specific for the AeG Personal Assistant. Please note that the AeG Personal Assistant client is only one type of possible user client systems and that other requesters, for example other web services, are capable of using the AeG Infrastructure for service discovery, composition and execution.

The AeG Personal Assistant client provides the following functionalities:

- *User login and profile management*: AeG PA client manages user logins and user profiles. AeG PA client will allow to log into the system and use the functionalities provided by the AeG Platform. Logging is not mandatory, but some functionality may only be provided in limited ways without user authentication. The user profile stores personal data and outputs of any previous activities that are supposed to be reused in any subsequent service requests, so user does not have to enter every single piece of information again. Additional user profiles will therefore help to automatically complete information required to apply for certain public services. User profiles are formalized as the ontology with the data instances stored in a repository managed by the AeG PA client. The question which information will be stored in the user's profile is subject of upcoming specifications.

- *Life event/Goal discovery*: the AeG PA client provides a user interface, which will allow users to browse and select specific life events. For example, life events can be organized in multiple hierarchies and the user can navigate through these hierarchies to select his/her actual life event. The AeG PA client will also allow to specify full text queries for search in unstructured non-functional properties (i.e. name, text description, etc.) as well as in structured non-functional properties (i.e. information about the service provider) to search life events, goals or services. The AeG PA client will provide an interface to specify queries/preferences and to browse results. Any search procedure will be delegated to the AeG Discovery component.

- *Interface for the Scenario Execution Management*: To manage execution of the orchestrated scenarios, the AeG PA client will:

    a) Generate web user interfaces according to the semantic description of life events and services. This interface will allow to specify inputs for the goal preconditions in the goal discovery, and to specify inputs to execute or update the state of the scenario processes. Several types of user display interfaces will be generated with regards to the specific audience (standard Web browser, Kiosk, WAI etc.),

b) Visualise the scenario processes and inspect their state. This will include presentation of all information required for the invocation of services. As far as possible, the current state of a public service procedure will be communicated to the user and the AeG PA client will hence inform the respective user after approval or rejection by email or mobile phone messages,

c) Browse discovery results and allow to select a specific service when more than one service can achieve the requested goal (or sub-goal in the case that generic scenario is customized to a specific citizen request). During browsing the list of pre-defined life events, detailed information about the services will be displayed as well, to assist in selecting the relevant ones.

4.5.9 Interdependencies between components

In this chapter, we will present a sequence diagram to illustrate interactions between the AeG Personal Assistant client and the Discovery, Composition and Execution components of the AeG Infrastructure in order to compose and execute orchestrated scenarios for requested goals. The sequence diagram includes both traditional and web services.

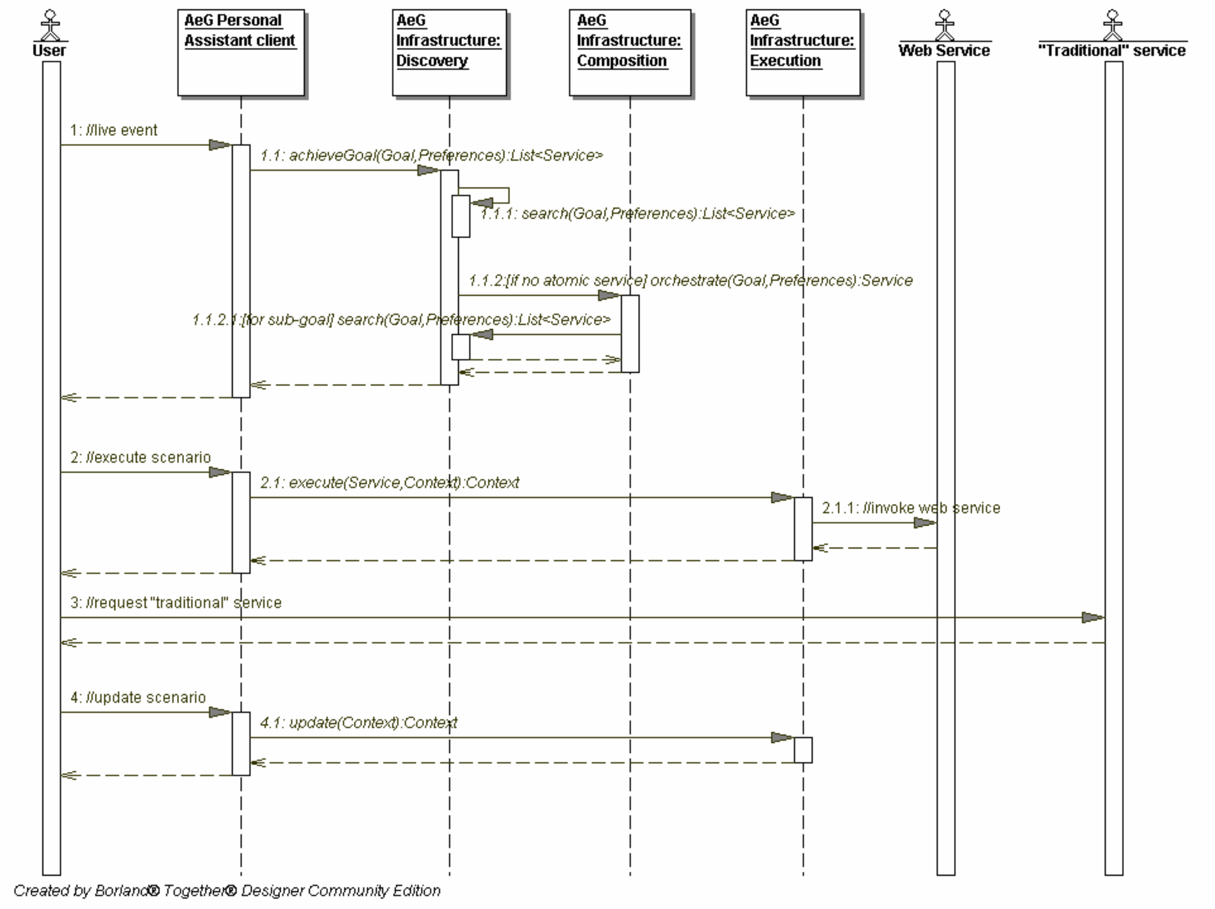
User
AeG Personal Assistant client
AeG Infrastructure: Discovery
AeG Infrastructure: Composition
AeG Infrastructure: Execution
Web Service
"Traditional" service
1: //live event
1.1: achieveGoal(Goal,Preferences):List<Service>
1.1.1: search(Goal,Preferences):List<Service>
1.1.2:[if no atomic service] orchestrate(Goal,Preferences):Service
1.1.2.1:[for sub-goal] search(Goal,Preferences):List<Service>
2: //execute scenario
2.1: execute(Service,Context):Context
2.1.1: //invoke web service
3: //request "traditional" service
4: //update scenario
4.1: update(Context):Context
Created by Borland® Together® Designer Community Edition

Figure 11 Sequence diagram "User interaction"

The description of interactions contains some implementation details (i.e. signatures of main methods in module interfaces), but all of these details are subject to change and are included for the sake of completeness only.

Activity	Description
1.	A citizen uses the web interface of the AeG Personal Assistant client to browse life event (LE) categories and to select the category for his/her LE. If some goals of the LE category have some preconditions, the AeG Personal Assistant generates a form where the user can enter values for all precondition variables which cannot be resolved from the user's profile. After evaluation of the preconditions, the goals are concatenated and send to the Discovery component of the AeG infrastructure module. Citizen can additionally specify some preferences (non-functional properties) for filtering and reordering of the discovered services either for this request or in the user profile.
1.1 1.1.1	The Discovery component will match semantic descriptions of the requested goal with the service profiles and find those services which are able to achieve this goal. Preferences specified by the requester are then used to additionally filter or reorder discovered services.
1.1.2	If the Discovery component cannot find any service capable to achieve a requested goal (either atomic or composed), the goal description is delegated to the Composition component which will try to orchestrate existing services (semi-)automatically into the new scenario. The orchestrated scenario is then registered as a new composed service, so that subsequent requests to achieve the same goal will not invoke the Composition component again.
1.1.2.1	During the planning phase, the Composition component can decompose requested goals into sub-goals and may use the Discovery component to find services achieving these sub-goals.
2. 2.1	The AeG Personal assistant visualizes an orchestrated scenario and may ask the citizen to enter input information specified for the composed service representing a scenario. Inputs are sent to the Execution component and stored in the execution context container.
2.1.1	In case the process activity invokes a web service, the Execution component will use grounding mechanisms specified for the web service to translate inputs and outputs.
3.	For a "traditional" service, the AeG Execution component will suspend the scenario execution and send a semantic description of the service to the AeG Personal Assistant client. AeG Personal Assistant client will present instructions of how to request this service and the citizen will follow these instructions and contact or visit the public authority in person. Instructions presented to him/her are extracted from non-functional properties of the service (i.e. information about the provider) and from the description of service inputs and outputs.
4. 4.1	The citizen will use the AeG Personal Assistant client web interface to provide information about the state or the result of the "traditional" services. After updating the context data, the Execution component may restore execution of the scenario process and continue with any subsequent activities.

5 Physical architecture

In the previous sections we have outlined general architecture of the Access-eGov System, without specifying any details on the implementation of such a system. In medium- and large-scale software projects there is a need to build on existing standards and to reuse as many components as possible. When it comes to architectural specifications, we have identified two main areas in which we can reuse already existing projects: the knowledge formalism and the platform itself.

5.1 Security view

On the physical layer, security features in Access-eGov mainly encompass a selection of technologies related to Web service security. Since the Access-eGov platform will be operating on a service oriented architecture network, XML-messages will be dealt with in accordance to international standards by W3C and OASIS. Access-eGov will make use of a selected subset of WS-Security mechanisms on application level. The Web Services Security (WSS) specification from OASIS defines details of how to apply XML signature and XML encryption concepts in SOAP messaging. WSS relies on XMLDS and XML encryption for low level details and defines a higher-level syntax to wrap security information inside SOAP messages providing three main security features: Message Integrity, User Authentication and Confidentiality.

In combination with SAML (Security Assertion Mark-up Language), Access-eGov will offer an optional Single-Sign-On mechanism available to all participating public agencies. The security tokens will be issued by the platform after a user has been successfully authenticated via a Personal Assistant instance.

Before forwarding a service-request to the respective public service provider’s SOAP-server, XML-firewalls will check each single SOAP-message for its XML encryption and signature part in order to disburden the service provider’s application servers and to reduce network traffic in the back-office sector. Additionally, traffic between participating parties in Access-eGov will be encrypted via SSL technology on network level.

Further information about the actual protocol stack will be provided in detail in the upcoming documents on security features and on component functionality in Access-eGov.

5.2 Data view

5.2.1 Candidates of knowledge formalisms

As mentioned above, domain knowledge has to be kept persistent, the storage itself should be decoupled from the rest of the platform, and therefore the best solution will be to store this kind of data in ontologies. There is a couple of candidates for these semantic formalisms, as already explained in the State-of-the-Art report [D2.1] Therefore there is no need to describe those formalisms again in greater detail. The following paragraphs will be mainly concerned with implications the user requirements have on preferring one formalism to the others.

The relevant user requirements (UREQ) are:

- R2.1.1 (multilingual support), R2.2.2 (possibility to store step-by-step descriptions), R2.3.1 (possibility to effectively search the stored knowledge), R2.2.3 (some sort of mapping between user's goal and offered services or chain of services)
- Additionally, the general requirements (GREQ) in chapter 2.3.1 apply, extended by the specific requirements for the semantic layer (SREQ) in chapter 2.4.

5.2.1.1 OWL-S

OWL-S is an OWL ontology for semantic description of Web Services. The structure of the OWL-S consists of a service profile for service discovering, a process model which supports composition of services, and a service grounding, which associate profile and process concepts with the underlying service interfaces.

	Pro	**Con**
UREQ	• Suitable for all UREQs. • Extension of the standardized OWL.	• R2.2.3 has no clean solution in OWL-S as there is no clear distinction between the users' goals and the service offerings. • No mapping for traditional services • Mediation is not part of the formalism • Defines orchestration, no choreography
GREQ	• Best choice as far as interoperability is concerned, because lots of applications are built around OWL/(-S)	• OWL-S recommends the combination of various rule languages (i.e. SWRL, DRS or KIF) with OWL to specify conditions. Combinations with SWRL or DRS lead to undecidability or leave semantics open. Furthermore, it is not clear how OWL and KIF descriptions interact (KIF syntax is treated as a string.
SREQ	Good integration of process modelling, suitable	

5.2.1.2 WSMO

The Web Service Modelling Ontology (WSMO) is a conceptual model for describing semantic Web Services. WSMO consists of four major components: ontologies, goals, Web Services and mediators.

	Pro	**Con**
UREQ	• Suitable for all UREQs. • Possibility of Access-eGov influence on specification process, because parts of	• No mapping for traditional (non web service)-services • Current orchestration model is

	the specification are not finished • Already contains the concept of mediation (between ontologies (oo), goals (gg), goals and services (wg) and services (wx).	too abstract and is not suitable for the interaction with the citizen (specification is not finished)
GREQ	• Mapping to OWL-S is formalised • Clean design • No add-on character as in the OWL-S case	• Not much real-world usage, yet • Not many WSMO-external projects use it, yet
SREQ	Suitable	

5.2.1.3 WSDL-S

WSDL-S is a small set of proposed extensions to Web Service Description Language (WSDL) by which semantic annotations may be associated with WSDL elements.

	Pro	**Con**
UREQ	• Suitable for all UREQs. • Possibility of Access-eGov influence on specification process, because parts of the specification are not finished.	• Neither orchestration nor choreography formalized • Mediation is not formalized • No mapping for traditional (non web service)-services
GREQ	• Simple bottom-up approach, agnostic to the domain ontology language.	• Not much real-world usage, yet
SREQ	Basically suitable	

5.2.1.4 BPEL4WS

BPEL4WS (Business Process Execution Language for Web Services) is a specification that models the behaviour of Web Services in a business process interaction. It represents a convergence of the ideas in the XLANG and WSFL specifications. Both XLANG and WSFL are superseded by the BPEL4WS specification and therefore they are not presented separately in this report. It is based on the XML grammar which describes the control logic required to coordinate Web Services participating in a process flow. An orchestration engine can interpret this grammar so it can coordinate activities in the process.

	Pro	**Con**
UREQ	• Specialized language • Offers more functionality than the requirements demand	• Main focus on orchestration and choreography, no semantic annotations in the model
GREQ		• Not much real-world usage, yet
SREQ	Partially suitable, it could be used to describe workflows, heavily depends on the outcome of the WSMO-Orchestration discussions	

5.2.1.5 Conclusions

This closer look at possible candidates for knowledge representation formalism has shown that not all four candidates are (re-) usable in our context. WSDL-S is not at all what the project needs, BPEL4WS is rather a workflow composition / description formalism.

The only full-featured packages left for consideration are WSMO and OWL-S. While OWL-S could prove to be a better decision in terms of the community support, WSMO seems to be a technically more advanced option. Its very good design, with the inclusion of mediators and the distinction between goals and capabilities, combined with a vivid professional community, make it the best choice for the Access-eGov infrastructure at this point of time.

5.3 Structure view

5.3.1 Candidates for frameworks

If we examine the candidates for the whole frameworks, we need to focus on slightly different requirements. First of all, requirements for the semantic layer (SREQ) do not apply anymore. We need however, to take into account all the filtered user requirements and general requirements (UREQ and GREQ in Chapter 2).

Another important part is the relationship to WSMO. The chosen framework will have to support WSMO as we have already selected the WSMO as our primary ontology framework of choice. The existence of formal mappings to OWL-S (even including implementations) yields room for an objective look at the candidates, though.

Already the State-of-the-Art report [D2.1] identified candidate technologies for the generic task of supporting semantic web services. The following section will compare those technologies and find the best suiting candidate for the Access-eGov technology platform (WREQ).

5.3.1.1 METEOR-S

METEOR-S project proposes the application of semantics to existing Web Service technologies. In particular the project endeavours to define and support the complete lifecycle of Semantic Web Service processes. The project extends WSDL to support the development of Semantic Web Services using semantic annotation from additional type systems such as WSMO and OWL ontologies.

	Pro	**Con**
UREQ	• Suitable for all UREQs.	
GREQ	• Interoperability between OWL-S and WSMO through mapping • Robust design • Questionable project activity, no recent website updates, no new downloads since mid-June 2005	• No real-world usage yet, purely academic
WREQ	Supported through mapping ➔ lack of native support	

5.3.1.2 WSMX

Web Service Execution Environment (WSMX) is an execution environment which enables discovery, selection, mediation, and invocation of Semantic Web Services. WSMX is based on the conceptual model provided by WSMO, being at the same time a reference implementation of it. It is the scope of WSMX to provide a test bed for WSMO and to prove its viability as a mean to achieve dynamic interoperability of Semantic Web Services.

	Pro	Con
UREQ	• Suitable for all UREQs	
GREQ	• Very advanced and flexible design • Interoperability between WSMO and OWL-S through mapping • Very agile project development • Very good codebase • Builds up on existing standards • Many EU projects are starting to use WSMO/WSMX	• Not all proposed components are implemented yet • Grounding remains an open question
WREQ	Reference implementation of WSMO ➔ good choice	

5.3.1.3 IRS III

The Internet Reasoning Service (IRS) is a framework for Semantic Web Services that supports the publication, location composition and execution of Web Services based on their semantic descriptions. IRS supports the conceptual model defined by WSMO and also provides mappings for service descriptions provided in OWL-S.

	Pro	Con
UREQ	• Suitable for all UREQs	
GREQ	• Interoperability to WSMX • “One-click publishing”, automatic creation of wrapper classes for LISP and Java code • Capability-driven service execution • Extensible / programmable ➔ plug-in concept • Builds up on existing standards	• No direct support for OWL-S known currently, only on ontology level through import functionality • Binding between goal and service via mediators is dynamic but binding between goal and mediator is static (formed at design time) • Codebase of the server in LISP, risky because not many LISP developers on project • Grounding issues are settled
WREQ	Reference implementation of WSMO ➔ good choice	

5.3.2 Access-eGov architecture expressed in WSMX

The overall architecture of the WSMX system is comprised of components that are described in detail in the WSMX deliverable D13.4, "WSXM Architecture.[8]

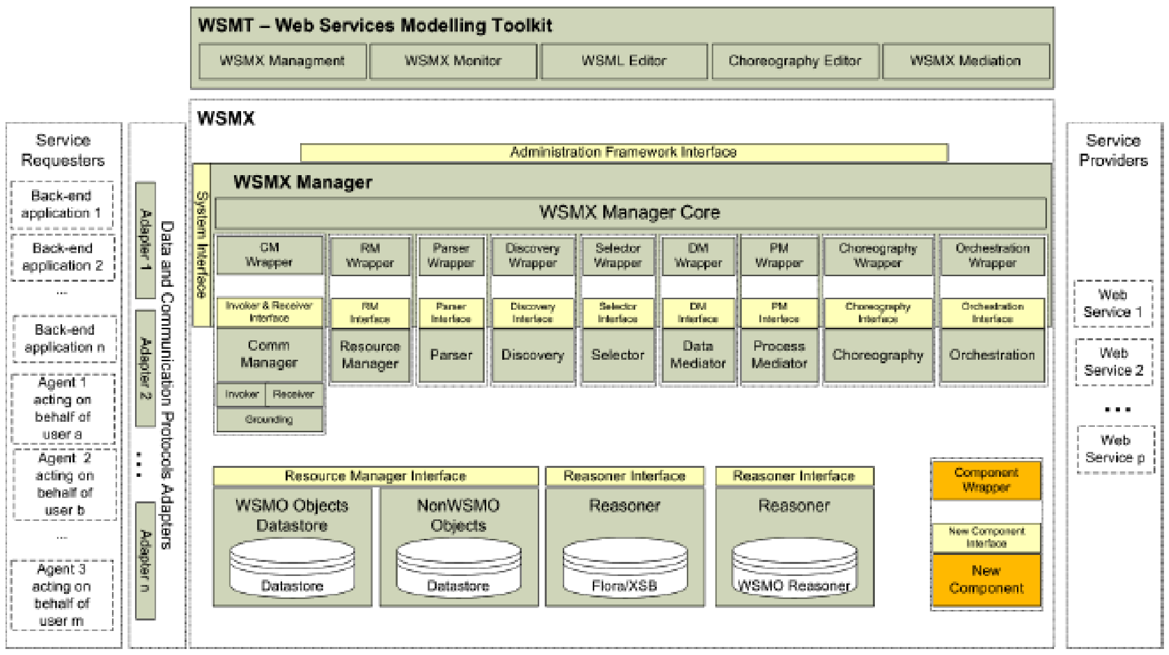

Figure 12 WSMX components

Access-eGov "native"	WSMX Component
AeG Infrastructure Core components	**Core Component** The Core Component is the central part of WSMX, and all the other components are managed by it. All interactions between components will be coordinated through the Core Component. The business logic of the system, the events engine, the internal workflow engine, the distributed components loading, etc. will all be subcomponents of the Core Component. At this stage the Core Component is the central module of WSMX. In the future, in order to ensure increased reliability of WSMX, a clustering mechanism for distributed Core Components might be developed. For resource intensive operations, many instances of the Core Components can be instantiated on several machines to create a WSMX cluster. **Parser Interface** The Parser checks if the syntax of received WSML descriptions is correct. Only if correct, the descriptions are parsed into WSMO4j data object used as the internal data representation for WSMO. **Communication Manager**

[8] http://www.wsmo.org/TR/d13/d13.4/v0.3/20051012/20051012_d13_4.pdf

	The Communication Manager is responsible for dealing with the protocols for sending and receiving messages to and from WSMX. Its external behaviour is accessed through the Invoker and Receiver interfaces. The WSMX Receiver interface expects the contents of all messages it receives to be expressed in WSML. Each WSML messages may represent a goal to be achieved or be a message corresponding to a choreography or orchestration instance that already exists. The Communication Manager accepts the message and handles any transport and security protocols used by the message sender. The execution semantics of WSMX determine how the WSML message should be handled based on the defined execution semantics of the system.
	Conclusions - WSMX Core Component can be extended to manage new modules which will implement functionality not defined in the WSMX (i.e. composition module for generic scenarios) - Depending on the definition of the process model for generic scenarios, Parser interface will need to be extended for the new syntax constructions. - WSMX Receiver has to be extended to support AeG Execution and Composition interfaces.
AeG Infrastructure Discovery	**Service Discovery** The WSMX Discovery component is concerned with finding Web service descriptions that match the goal specified by the service requester. The intent is that the WSMO description of the goal a user wishes to achieve (described in terms of a desired capability with preconditions, assumptions, effects and post-conditions) is matched to the WSMO description of Web services known to WSMX (described in terms of offered capabilities). The discovery component returns a (possibly empty) list of Web service descriptions. **Non-functional Selector** The Non-functional Selector is a component used to select the most suitable service from a list of matching services matched by discovery. For example, a service requester may define preference for the selection of the most suitable discovered Web service. If discovery results in more than one service that satisfies the goal, the selection interface is used to chose one based on specified preferences. Selection does not involve making an invocation on the service. **Negotiation (Functional Selector)** The Functional Selector is used to select the most suitable service from a list of matching services where the services must be actually invoked as part of the selection procedure. **Reasoner Interface** There is no agreement on a stable Reasoner interface so far. Once

	work on WSMO reasoner implementation will finalize, its interface will become standardized through WSMX info model.
	Conclusions According to the WSMX specification, Service Discovery, Functional Selector, Non-functional Selector, Reasoner interface and Reasoner implementation will together almost entirely cover the functionality required from the AeG Discovery component. This will include service keyword search and simple and rich semantic discovery. Full-text search of goals/life events is not covered by WSMX and has to be implemented in the stand-alone sub-component.
AeG Infrastructure Composition	**Conclusions** WSMX specifies a model for orchestration and choreography which is based on the state machines. However this model is too abstract and is not intended for example for the interaction with human actors. AeG Composition will be based on the structured process model of the generic scenarios which can be used to guide citizen to achieve specific goal (see Functional Data view chapter 4.4.5)
AeG Infrastructure Execution	**Invoker** The Invoker is used by the execution semantics of WSMX when a Web service needs to be invoked. The invoker receives the WSMO description of the service and the data that the service expects to receive. It is responsible for making the actual invocation of an operation on a service. In the majority of existing Web service implementations, this means ensuring that the semantic description of both the data and the behaviour of the Web service are grounded to the corresponding WSDL descriptions. It is anticipated that a separate grounding component to work with the invoker will be required in future versions of WSMX.
	Conclusions Invoker sub-component can be used to invoke web services with the grounding of the semantic description data. Execution engine, which will manage the instances of the scenario processes has to be implemented according to the designed process model, existing WSMX components designed for the orchestration and choreography cannot be used directly (see AeG Composition comments)
AeG Infrastructure Mediation	**Data Mediator** WSMX Data Mediator component has the role of reconciling the data heterogeneity problems that can appear during discovery, composition, selection or invocation of Web Services. This interface is dedicated for the runtime phase mediation process. The run-time component implementing this interface has the role of retrieving from storage the already-created mappings, to transform them into rules, and finally to execute them against the specified incoming instances (input) in order to obtain the target instances

	(output). Since the mappings represent the connection point between the two sub-components (design-time and run-time) one of the dependencies for the run-time component relates to the mapping storage level. Another crucial dependency relates to the reasoning system used for executing the rules in the final stage of the mediation process.
	Conclusions WSMX Data Mediator component covers entirely data mediation and ontology merging requirements defined for the AeG Mediation component, this includes support for the design of ontology mappings implemented in the WSMT
AeG Infrastructure Data repositories	**Resource Manager** The Resource Manager is the interface for WSMX persistent storage. The component implementing this interface is responsible for storing every data item WSMX uses. The WSMO API provides a set of Java interfaces that can be used to represent the domain model defined by WSMO. WSMO4j provides both the API itself and a reference implementation but it is not a pre-requisite that implementations of the Resource Manager use WSMO4j. Currently WSMX defines interfaces for six repositories. Four of these repositories correspond to the top level concept of WSMO, i.e. Web services, ontologies, goals, and mediators. The fifth repository is used by WSMX for non-WSMO data items, e.g. events and messages. Finally the sixth repository stores WSDL documents used to ground WSMO service descriptions to SOAP or SOAP/HTTP. In the longer term the WSMX working group recognises that ebXML and UDDI repositories can be used for WSMX data persistence.
	Conclusions Resource Manager has to be extended with interfaces for the new data objects defined for the process model and context of the process execution. AeG Personal Assistant client and AeG Annotation Services can leverage this component, but they will require additional types of repositories for example for the user profiles and security data.
AeG Annotation Services	**Web Service Modelling Toolkit Interface** The Web Services Modelling Toolkit (WSMT) provides user interface tools for WSMX management and monitoring including a WSML editor.

6 Risks relevant to the proposed architecture

6.1 Make or buy

The decision whether to generate software components from scratch or to buy commercially available software components off the shelf and tailor them, is a very common managerial decision for most IT-dependent businesses. Sufficient reasons do exist to justify both possible options and those demand thorough investigation.

Until the onset of commercial off-the-shelf products for professional use, most new IT-investments have been done in-house, mostly because systems were too complex to be easily transferable. Over the last decades, this decision has more and more been answered with "buy", since the rapid spread of certain software products led to an ever growing supply of even more ready-made, commercial or free, closed or open source components related or supporting those commercial off-the-shelf products.

In Access-eGov, project decisions may not be strictly based on budgetary considerations. It would be possible to opt for creating every single component within the project from scratch. Given the current situation on the third party component market however, we can locate a great choice of components and program libraries which are ready for reuse.

Most Government off-the-shelf products are only available to public authorities and therefore not applicable for reuse in Access-eGov. This means that the consortium can only rely on software components that are available on the free market.

Although software reuse cannot be called free of risks, developing software from scratch has proven more risk-pregnant over the last couple of years [RED06]. For that reason, the Access-eGov consortium has decided to rely on using already manufactured software components in order to achieve the project goals.

6.2 General risks

Risk is commonly defined as a measure of the probability and severity of negative effects. Software architectural risk can then be defined as a measure of the probability and severity of adverse effects that the overall software architecture decisions entail for the development of software, ultimately resulting in a mismatch between its intended functions and the actual system performance.

Research studies in the field of Software Engineering conducted by the Carnegie Mellon University confirmed that about 80% of all possible risks related to project failure were attributed to contradictory requirements and unsatisfying architecture and design. In addition, the Management Process proved to be critically important in meeting the previously defined development requirements.

The need to manage both technical and non-technical (cost and schedule) risks increases with system complexity. In Europe, the number of failed software projects almost equals the number of projects that could successfully be finished on time with approximately 40–50% of all IT projects undertaken [LL06]. Over the years this percentage gradually increased from initially only 29% of failed software projects in 1985. This has happened in spite of promising new technologies, innovative methods and tools, and different management methods.

Based on year-long experiences in the field of Software Project Management, a number of recognized practitioners list the following items as top 10 software risks:

1. Personnel Shortfalls
2. Unrealistic schedules and budgets
3. Developing the wrong functions and properties
4. Developing the wrong user interface
5. Gold-plating
6. Continuing stream of requirements changes
7. Shortfalls in externally furnished components
8. Shortfalls in externally performed tasks
9. Real-time performance shortfalls
10. Straining computer-science capabilities

6.3 Project specific risks

Being situated in the especially risk-threatened group of multi-national, medium-scale software projects, the Access-eGov project consortium will have to address all risks emerging from technically-related aspects.

Especially, the consequences of the above mentioned risks have to be considered:

1) **Personnel Shortfalls**

   Although this project is only of medium size, it still is susceptible to the usual problems related with fluctuation in key personnel, especially taking into account the complexity of the used software components. A gap, caused by the retirement of a key developer with good WSMX know-how, could possibly be hard to be bridged.

   When selecting a platform like WSMX, the level of know-how already present is hard to measure. Not many developers will have in-depth knowledge of the technology yet. Therefore a key requirement for developers is the will and ability for permanent training. Effort will be spent to distribute WSMX know-how among persons involved in the development-oriented tasks of the project to reach a certain degree of redundancy and replaceability.

2) **Unrealistic schedules and budgets**

   This project has put lots of effort into drafting a realistic schedule. Some imponderabilities remain however. WSMX key components could, for example, lag behind schedule and it could turn out that it would be too time consuming to implement them in the scope of Access-eGov. In order to minimise this risk, progress of WSMX and its components will be closely followed to notice any potential problem in its early stage in order to evaluate its possible impact on the project and to plan appropriate remedial actions.

3) **Developing wrong functions and properties**

   Dealing with users is often a tricky task for developers. Users and developers tend to think in different terms, which leave many opportunities for misunderstandings. If such misunderstandings happen, and are not caught before implementation, the project could indeed develop a product with wrong or missing functions and / or wrong properties. In order to minimize this risk, user partners are involved in continuous communication with developer partners.

4) **Developing the wrong user interface**

   Not much of a risk for Access-eGov, as following guidelines and user requirements (together with effort spent on establishing live communication with user partners) will produce an adequate user interface.

5) **Gold-plating**

   Peer review of partners' work can minimize this risk.

6) **Continuing stream of requirements changes**

   This risk does not apply to this project, as the pilot projects have been formulated right at the beginning and are not about to change rapidly. Small changes in the requirements are anticipated, however, and developers are ready to incorporate them.

7) **Shortfalls in externally furnished components**

   Since Access-eGov has chosen WSMX as a basic technology for the project platform, this risk has already been bypassed by the project consortium. As WSMX is an open source project, the complete source code is available online and may at all times be tailored according to the needs arising over project time.

   Furthermore, the WSMX project is maintained by a number of prestigious institutions from various fields (research, private-sector companies and public agencies), and will unlikely lose developmental support from one day to another. Therefore, WSMX being a technology suite backed by one developing consortium is not supposed to imply any possible risk or threat to successfully completing the Access-eGov project work in terms of this risk.

8) **Shortfalls in externally performed tasks**

   There are no important tasks related to software engineering outside of the consortium.

9) **Real-time performance shortfalls**

   While some consideration has to be put into performance issues, the main focus is on implementing a technology platform using results gathered in research. Access-eGov will have to support the pilot projects, but will not have to broker between thousands of services for millions of users, at least not in the scope of this project. The architecture is flexible enough (due to design decisions we made) to be extremely scalable, though.

10) **Straining computer-science capabilities**

Increasing global research effort is being put into all the fields, also addressed by Access-eGov. No virgin soil is entered, therefore the project will stay well within the limits of current computer science.

According to [DHS06] most possible risks to software projects can be bypassed by following the minimum guidelines on Acceptable Secure Software Engineering Practices. In the combined Architecture/Design phase these practices consist of guidelines in order to:

a) "*Isolate all trusted/privileged functions*"

    - By nature, every function within the Access-eGov system will have to be authenticated prior to being used by other parties. Details of this procedure will be outlined in the Deliverable 3.2 ("Functional Description").

b) "*Avoid using high-risk services, protocols, and technologies*"

    - Since Access-eGov has chosen to use proven standard protocols and to integrate already existing electronic eGov services, no high-risk technology is being used in the software architecture.

c) "*Do not use insecure or untrustworthy components unless they can be effectively isolated and constrained to prevent their negative impact on the rest of the system*"

    - Components added by third parties and used within Access-eGov (e.g. WSMX technology) have proven to be secure in terms of practical operation over a reasonable period of time, and therefore cannot be categorized as untrustworthy. Nonetheless, executable legacy software used by user partners and loosely integrated to the system will have to be audited for possible impacts on the platform integrity. This work will be executed by the respective user partner and the test team(s).

d) "*Include only necessary functionality in from-scratch components. Remove or isolate unnecessary functions in acquired/reused components*"

    - Currently, WSMX consists to a large extent of functionality necessary to perform basic operations. Functionality reuse can be regulated generating stricter API implementations. Risks emerging from too generous XML-API implementations will be addressed in the design phase and in Deliverable 3.2 ("Functional Description").

e) "*Perform comparative security assessments of different assembly/integration options using different combinations of candidate components before committing to any component*"

    - These considerations have already been conducted while completing this document, namely Chapter 5.

These guidelines have been adhered to while outlining the system architecture. Therefore, the occurrence probability of any type of remaining risk that will emerge from architectural decisions seems to be low.

All other management-related risks will be addressed by the internal auditing procedures as set out in the Project Quality Plan (Deliverable 1.3).

Pending technology-related risks will be dealt with either in the upcoming Deliverable 3.2 (“Functional Description”) or during implementation phase by the developers.

7 Conclusions

In this document we outlined the overall architecture and defined all necessary components that are considered to be essential at the time of writing the report. Based on the requirements that could be gathered during interviews with all user partners of the project and based on the pilot descriptions as defined in the Technical Annex (DoW) to the contract, the required system functionality was the leading paradigm for the architecture.

After researching the most adequate networking architectures, we found out that a Service oriented approach in combination with a peer-to-peer-overlay-network would suit our project mission best.

On the conceptual architecture level we figured out the core functionality that should be provided by the system, namely from the information provider and consumer side of the system. There will be at least three different types of ontologies within the Access-eGov system: life events ontology, Service profile ontology and Access-eGov Domain ontology.

On the logical architecture level, the functionality required from the platform components, the Personal Assistant and the Annotation services could be detected using a couple of central Use Cases for service discovery, composition and execution. Regarding data, the minimum complexity of the semantic system and the ontologies was put down.

On the physical level, we tried to find counterparts for each Access-eGov component in the WSMX platform that shall provide the basis for our work.

Security views on each architectural level outlined the scope of the joint security architecture in Access-eGov.

Finally, all possible risks that may arise from architectural decisions have been pointed out. We thereby concluded that, if at all, we expect only minimal risks to emerge from these decisions.

8 Annex

8.1 Client-server architecture[9]

This chapter describes the basic concepts of client/server architecture, describes the two tier and three tier architectures and analyzes their respective benefits and limitations.

DEFINITION

The term client/server was first used in the 1980s in reference to personal computers (PCs) on a network. Though, client and server are software and not hardware entities. In its most fundamental form, client/server involves a software entity (client) making a specific request which is fulfilled by another software entity (server). Figure 13 illustrates the client/server exchange. The client process sends a request to the server. The server interprets the message and then attempts to fulfil the request. In order to fulfil the request, the server may have to refer to a knowledge source (database), process data (perform calculations), control a peripheral, or make an additional request of another server. In many architectures, a client can make requests of multiple servers and a server can service multiple clients.

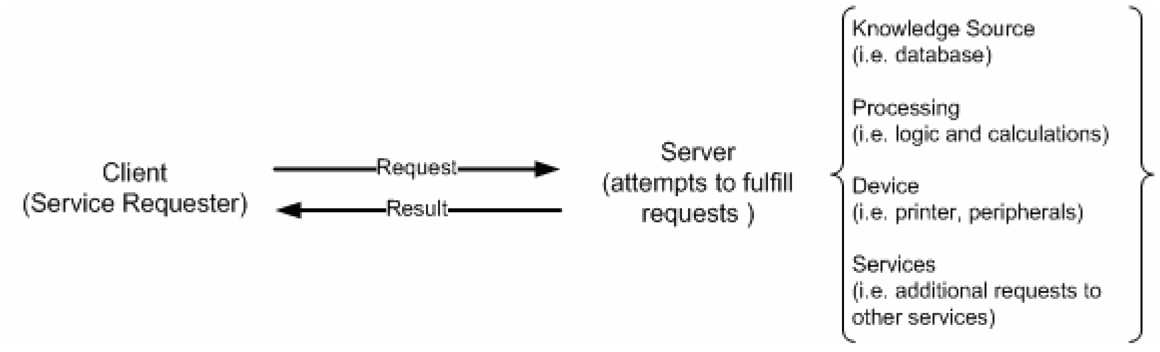

Figure 13 Client/server transactions

The relationship between client and server is a command/control relationship. In any given exchange, the client initiates the request and the server responds accordingly. A server cannot initiate dialog with clients. Since the client and server are software entities they can be located on any appropriate hardware. A client process, for instance, could be resident on a network server hardware, and request data from a server process running on another (server) hardware or even on a PC. In another scenario, the client and server processes can be located on the same physical hardware box. In fact, in the prototyping stage, a developer may choose to have both the presentation client and the database server on the same PC hardware. The server can later be migrated (distributed) to a larger system for further pre-production testing after the bulk of the application logic and data structure development is complete.

A distributed application consists of separate parts that execute on different nodes of the network and cooperate in order to achieve a common goal. The supporting infrastructure should also render the inherent complexity of distributed processing invisible to the end-user. The client in a client/server architecture does not have to sport a graphical user interface (e.g. for electronic mail delivery), however, the mass-commercialization of client/server has come about in large part due to the proliferation of GUI clients. Some client/server systems support highly specific functions such as print spooling (i.e. network print queues) or presentation services (i.e. X-Window).

[9] abridged from Gallaugher J., Ramanathan S., “The Critical Choice of Client Server Architecture: A Comparison of Two and Three Tier Systems”, 1996, http://www2.bc.edu/~gallaugh/research/ism95/cccsa.html

ARCHITECTURE TYPES

The vast majority of end user applications consist of three components: presentation, processing, and data. The client/server architectures can be defined by how these components are split up among software entities and distributed on a network. There are a variety of ways for dividing these resources and implementing client/server architectures. In the following lines, the most popular forms of implementation of two tier and three tier client/server computing systems will be presented.

Two tier architecture

In this implementation, the three components of an application (presentation, processing, and data) are divided among two software entities (tiers): client application code and database server (Figure). A robust client application development language and a versatile mechanism for transmitting client requests to the server are essential for a two tier implementation. Presentation is handled exclusively by the client, processing is split between client and server, and data is stored on and accessed via the server. The PC client assumes the bulk of responsibility for application (functionality) logic with respect to the processing component, while the database engine - with its attendant integrity checks, query capabilities and central repository functions - handles data intensive tasks. In a data access topology, a data engine would process requests sent from the clients. Sending database queries from client to server requires a tight linkage between the two layers. To send the SQL queries the client must know the syntax of the server or have this translated via an API (Application Program Interface). It must also know the location of the server, how the data is organized, and how the data is named. The request may take advantage of logic stored and processed on the server which would centralize global tasks such as validation, data integrity, and security. Data returned to the client can be manipulated at the client level for further sub selection, business modelling, "what if" analysis, reporting, etc.

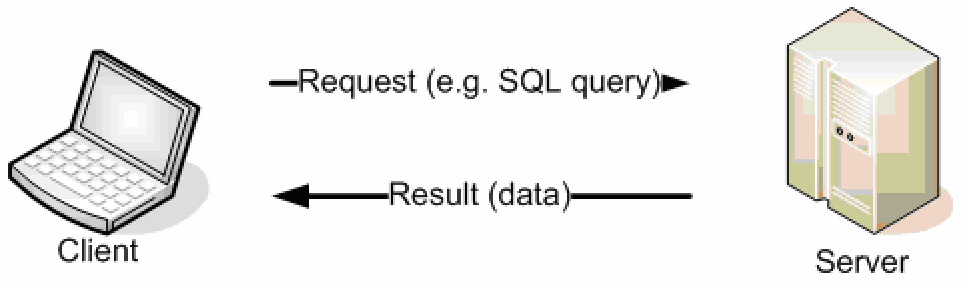

Figure 14: Data access topology for two tier architecture with most of the functional logic existing on the client side

The most compelling advantage of a two tier environment is application development speed. In most cases a two tier system can be developed in a small fraction of the time it would take to code a comparable but less flexible legacy system. Most two tier tools are also extremely robust. These environments support a variety of data structures, including a number of built in procedures and functions, and insulate developers from many of the more mundane aspects of programming such as memory management. Finally, these tools also lend themselves well to iterative prototyping and rapid application development (RAD) techniques, which can be used to ensure that the requirements of the users are accurately and completely met.

Two tier architectures work well in relatively homogeneous environments with fairly static business rules. This architecture is less suited for dispersed, heterogeneous environments with rapidly changing rules. As such, relatively few organizations are using two tier client/server architectures to provide cross-departmental or cross-platform enterprise-wide solutions.

Three tier architecture

The tree tier architecture (Figure) attempts to overcome some of the limitations of the two tier scheme by separating presentation, processing, and data into separate, distinct software entities (tiers). When calculations or data access is required by the presentation client, a call is made to a middle tier functionality server. This tier can perform calculations or can make requests as a client to additional servers. The middle tier servers are typically coded in a highly portable, non-proprietary language such as Java (in form of "servlets") or C++. Middle tier functionality servers may be multi-threaded and can be accessed by multiple clients, even those from separate applications.

Although three tier systems can be implemented using a variety of technologies, the calling mechanism from client to server in such as system is most typically the remote procedure call or RPC. Unlike most two tier implementations, the three tier presentation client is not required to "speak" languages of the back-end-systems just as SQL. As such, the organization, names, or even the overall structure of the back-end data can be changed without requiring changes to PC-based presentation clients linked via a network. Database structures can be modified without making changes to client software: data may thus be organized relationally or object-based. This added flexibility allows to access legacy data via wrapper components and simplifies the introduction of new database technologies.

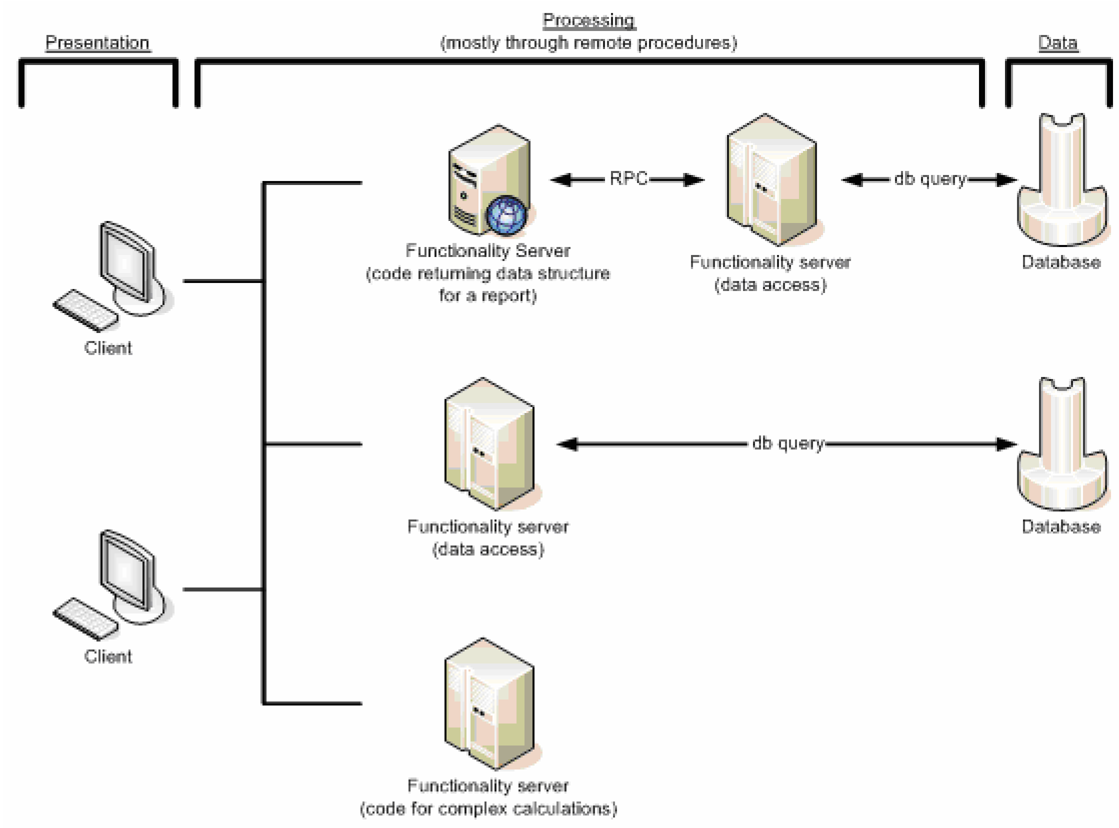

Figure 15: Three tier architecture with most of the logic processing handled by functionality servers

In addition to the openness stated above, several other advantages are presented by this architecture. Having separate software entities can allow for the parallel development of individual tiers by application specialists. Having experts focus on each of these three layers can increase the overall quality of the final application.

The three tier architecture also provides for more flexible resource allocation. Middle tier functionality servers are highly portable and can be dynamically allocated and shifted as the needs of the organization change. Network traffic can potentially be reduced by having functionality servers strip data to the precise structure required before distributing it to individual clients at the LAN level. Multiple server requests and complex data access can emanate from the middle tier instead of the client, further decreasing traffic. Also, since PC clients are now dedicated to just presentation, memory and disk storage requirements for PCs will potentially be reduced.

8.2 Advanced distributed components (J2EE)[10]

Sun Microsystems' J2EE (Java 2 Enterprise Edition) provides the context for a bunch of technologies with Enterprise JavaBeans (EJB) being the component-based architecture for building distributed business applications in the Java programming language. The EJB architecture includes the concepts of EJB server and EJB container.

The EJB server acts as a component execution system and provides the operational environment for applications using EJB components. It delivers all necessary services to support the EJB architecture. Yet there is no prescribed way to package EJB server software.

EJB components do not execute directly on top of the EJB server. An intermediate software component referred to as the EJB container runs within the EJB server environment, and in turn provides the operational environment for the components, called "beans" themselves. EJB containers are completely transparent to EJB applications, but play a critical role in supporting bean operation.

In order for enterprise beans to act as reusable software components, they do not have built-in dependencies on specific server or platform functionality. But several common types of server-side functionality have been "factored out" of the bean design, with responsibility for this functionality transferred to the container component. For example, containers are intended to take over responsibility for security, concurrency, transactions, swapping to secondary storage, and other services, allowing beans to be free of server dependencies, and to be optimized for business logic rather than service logic.

When a developer designs a new EJB component, writing the code that makes up the enterprise bean class is not enough in itself. The EJB programmer must also write two Java interfaces that will be used by helper classes. Up to a certain degree, this work is automatically executed by the development environment.

It is anticipated that EJB container software typically will be shipped with EJB server software, although the specification allows separation of these components. In addition to providing access to run-time services such as transactions and security, EJB containers are also expected to include a variety of tools necessary to support installation, operation, and management of enterprise beans. For example, tools are required to interpret the contents of EJB jar files, to generate database access for container-provided persistence, to monitor behaviour of running beans, and to implement security.

An EJB-based three tier programming model views a Web browser as the first tier, an application-enabled Web server as the second tier, and enterprise information resources as the third tier. In addition to EJB technology, Java servlet technology, JavaBeans technology, and Java Server Pages (JSP) technology are also implemented in this programming model. The following illustration shows the arrangement of tiers:

[10] abridged from Nordby K., "What are Enterprise JavaBeans components?", IBM 2001, http://www-128.ibm.com/developerworks/java/library/j-what-are-ejbs/part3/

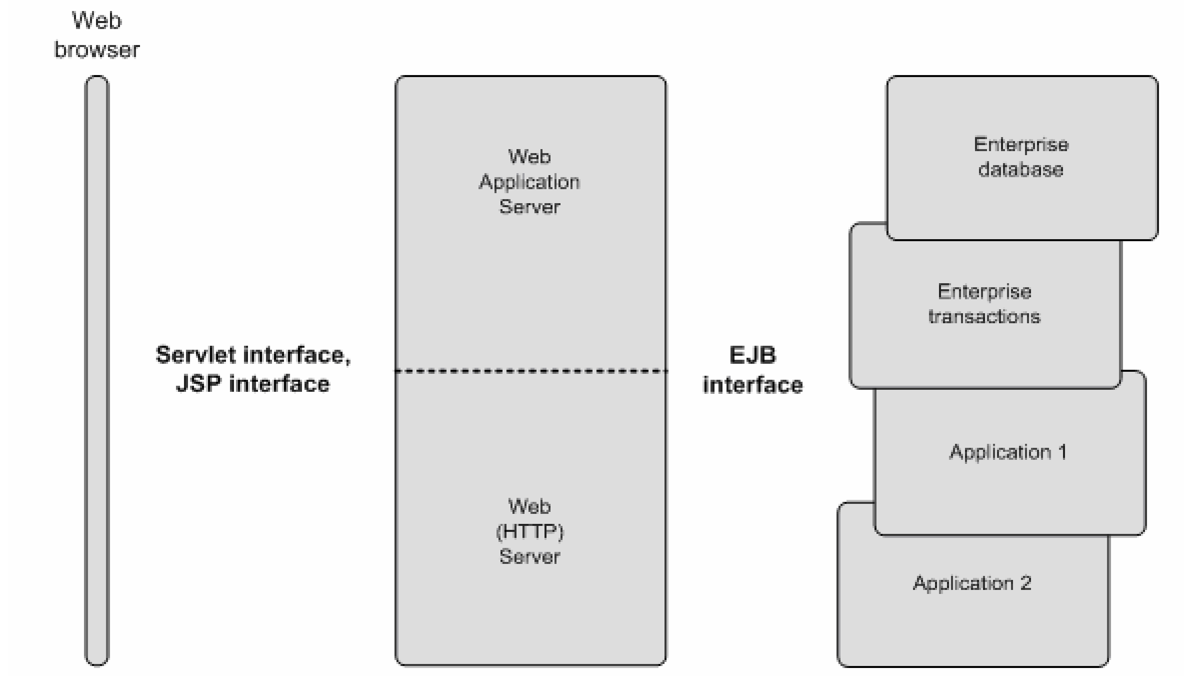

Figure 16: Three tier architecture as proposed by J2EE

Java servlets, beans, and server pages are not required elements of EJB application scenarios, but are supposed to work together with EJB components to provide a cohesive Java-based application environment. The J2EE environment assigns the following responsibilities to the participating Java components:

- Java servlets are assigned the role of application "controller"
- JSP pages handle presentation of data and user interface tasks
- Java beans play the role of "data encapsulator" storing intermediate results data
- EJB components provide the mechanism for accessing enterprise information resources

On the technical level, the EJB Transaction Model shows a number of limitations that we will not enter into details: transactions cannot manage locks according to the application's requirements, the set of values for the transaction attribute is very limiting, bean methods cannot be associated with several transaction attributes, bean methods cannot be dynamically associated with a particular transaction attribute, no support for asynchronous operations.

8.3 Service oriented architecture[11]

Technically, the term "service-oriented architecture" (SOA) refers to the design of a system, not to its implementation. SOA as an architectural style emphasizes implementation of components as modular services that can be discovered and used by clients. In considering the term service-oriented architecture, it is useful to review the key terms:

- An **architecture** is a formal description of a system, defining its purpose, functions, externally visible properties, and interfaces. It also includes the description of the system's internal components and their relationships, along with the principles governing its design, operation, and evolution.

[11] abridged from Srinivasan L., Treadwell J., "An Overview of SOA, Web Services and Grid Computing", Hewlett-Packard 2005, http://h71028.www7.hp.com/ERC/downloads/SOA-Grid-HP-WhitePaper.pdf

- A **service** is a software component that can be accessed via a network to provide functionality to a service requester.
- The term **service-oriented architecture** thus refers to a style of building reliable distributed systems that deliver functionality as services, with the additional emphasis on loose coupling between interacting services.

Services then generally have the following characteristics:

- Services may be individually useful, or they can be integrated – composed - to provide higher-level services. Among other benefits, this promotes re-use of existing functionality.
- Services communicate with their clients by exchanging XML-compliant messages: they are defined by the messages they can accept and the responses they can give.
- Services can participate in a workflow, where the order in which messages are sent and received affects the outcome of the operations performed by a service. This notion is defined as "service choreography".
- Services may be completely self-contained, or they may depend on the availability of other services, or on the existence of a resource such as a database. In the simplest case, a service might perform a calculation such as computing the cube root of a supplied number without needing to refer to any external resource, or it may have pre-loaded all the data that it needs for its lifetime. Conversely, a service that performs currency conversion would need real-time access to exchange-rate information in order to yield correct values.
- Services advertise details such as their capabilities, interfaces, policies, and supported communications protocols. Implementation details such as programming language and hosting platform are of no concern to clients, and are not revealed.

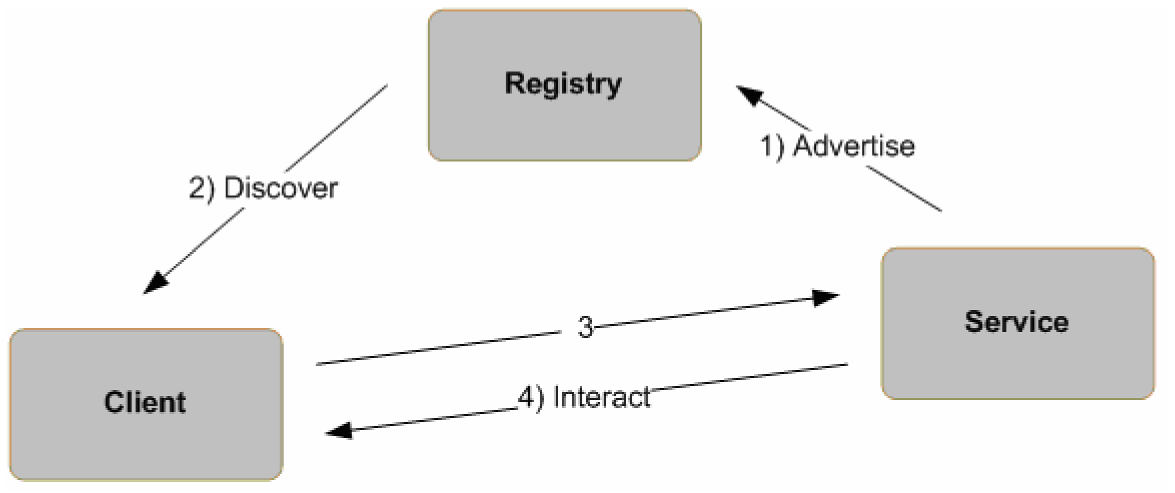

Figure 17: Service interaction in a service-oriented environment

Figure 17 illustrates a simple service interaction cycle, which begins with a service advertising itself through a well-known registry service (1). A potential client, which may or may not be another service, queries the registry (2) to search for a service that meets its needs. The registry returns a (possibly empty) list of suitable services, and the client selects one and passes a request message to it, using any mutually recognized XML-based protocol (3). In this example, the service responds (4) either with the result of the requested operation or with a fault message.

The illustration shows the simplest case, but in a real-world setting such as a governmental application the process may be significantly more complex. For example, a given service may support only the HTTPS protocol, be restricted to authorized users, require Kerberos authentication, offer different levels of performance to different users, or require payment for use. Services can provide such details in a variety of ways, and the client can use this information to make its selection. Some attributes, such as payment terms and guaranteed levels of service, may need to be established by a process of negotiation before the client can make use of the service it has selected. And, while this illustration shows a simple synchronous, bi-directional message exchange pattern, a variety of patterns are possible - for example, an interaction may be one-way, or the response may come not from the service to which the client sent the request, but from some other service that completed the transaction.

Figure shows a client engaged in a three-step transaction with several services, each of which might be capable of handling any part or all of the transaction. The service that handles Step 1 stores the details of the in-progress transaction in the database, and returns requested information to the client, along with a transaction identifier. The client might request confirmation from the user before passing the transaction identifier to another service, which uses it to retrieve the state information from the database and initiates Step 2. This service then updates the database and returns additional information to the client. Finally, the client passes the transaction identifier back to the third service with a request to complete the transaction.

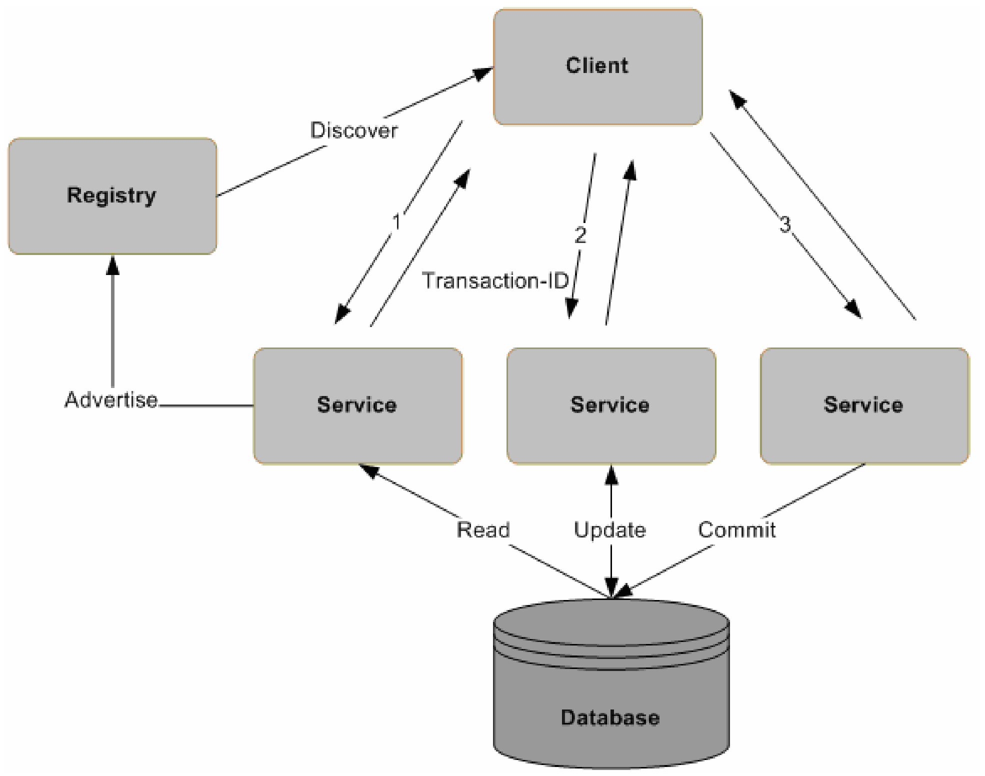

Figure 18: A multi-step client/service interaction

WEB SERVICES

Since service-orientation is an architectural style, Web services can be seen as the corresponding implementation technology. The two can be used together, and they frequently are, but don't need to be mutually dependent.

Although it is widely considered to be a distributed-computing solution, SOA can be applied to advantage in a single system, where services might be individual processes with well-defined interfaces that communicate using local channels, or in a self-contained cluster, where they might communicate across a high-speed interconnect.

Similarly, while Web services are well-suited as the basis for a service-oriented environment, there is nothing in their definition that requires them to embody the SOA principles. While statelessness is often held up as a key characteristic of Web services, there is no technical reason that they should be stateless - that would be a design choice dictated by the architectural style of the environment in which the service is intended to participate.

The key specifications used by Web services are:

- SOAP (originally Simple Object Access Protocol, but technically no longer an acronym) – an XML-based protocol for specifying envelope information, contents and processing information for a message.
- WSDL (Web Services Description Language) – an XML-based language used to describe the attributes, interfaces and other properties of a Web service. A WSDL document can be read by a potential client application to learn about the service and to retrieve meta-information.
- UDDI (Universal Description, Discovery, and Integration) – is a specification for a group of web-based registries to advertise information about Web services and their WSDL-described interfaces. UDDI specifies both interfaces for describing the service registries and also specifies how the service registries can be operated.

In order to integrate legacy systems into service oriented architectures, wrapper applications will need to be attached to them. The wrapper acts as an intermediary for the conventional application by exposing a Web service interface to service requesters and that takes XML files as inputs. These proxy interfaces are usually charged with listening for incoming SOAP calls, pre-processing the incoming XML messages and finally invoking the legacy applications. Wrapper applications therefore consist of three distinct parts. The first part maps the schema to callable methods. The second prepares XML data for the application and converts the data for further processing. The third one invokes the actual legacy application with the pre-processed input data using a protocol handler.

Adding Web service wrappers may affect the external and potentially the internal calling applications by exposing functionality through standard SOAP calls.

8.4 Peer-to-peer architecture[12]

Peer-to-Peer (P2P) is a class of applications that takes advantage of resources - storage, cycles, content, human presence - available at the edges of a network. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P design requirements commonly include independence from DNS and significant or total autonomy from central servers. Their implementations frequently involve the creation of abstract, virtual overlay networks with a structure independent of that of the underlying physical Internet.

Initially, the architecture has been popularized by mass-culture (music) file-sharing and highly parallel computing applications that scale in some instances to hundreds of thousands of nodes. P2P communities are now seeking to expand to more sophisticated applications as well as continuing to innovate in the area of large-scale autonomic system management.

Participants in peer-to-peer architectures simultaneously serve and receive files. Most P2P systems handle file sharing in a decentralized fashion. Some systems however fall back to a

[12] abridged from Foster I., Iamnitchi A., 2003, “On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing”, 2003, http://people.cs.uchicago.edu/~anda/papers/foster_grid_vs_p2p.pdf

client-server architecture for indexing and searching. There are thus two main approaches, which are named after the first popular systems that implemented them[13]:

- The “Napster” (also called hybrid or first-generation) approach: A centralized group of servers indexes filenames, and all queries must go through them.
- The “Gnutella” (called pure or second-generation) approach: No central indices are maintained; queries are broadcast to a node’s “neighbours” (which then broadcast them to all of their neighbours, and so on, up to a fixed number of steps, called the horizon).

Over time, P2P systems have evolved from first generation centralized structures (e.g., Napster index, SETI@home) to second generation flooding-based (e.g., Gnutella file retrieval) and then third generation systems based on distributed hash tables. First and second generation P2P collaborations have been characterized at the level of both individual nodes (behaviour, resources) and network properties (topological properties, scale, traffic), revealing not only general resilience but also unexpected emergent properties. Third generation systems have been characterized primarily via simulation studies rather than large-scale deployments and until today have not been popularized. Scalable autonomic management clearly has been achieved to a significant extent in P2P, albeit within specific narrow domains.

Today, large P2P communities exist: millions of simultaneous nodes in the case of file-sharing systems and several million total nodes in SETI@home for instance. In 2001, SETI@home handled up to 1-2 TB of data transfer per day.

P2P systems are successful for several reasons, including:

- Ease of deployment: Each user installs a single package that encompasses both client and server code; its initial configuration depends only on knowing a fixed index server or a single other installation; servers need not be continuously active.
- Ease of use: The server code is bundled with a user interface application to publish, search and retrieve content.
- Fault tolerance: Failure or unavailability of a single server (other than a central index) does not disable the system. It might render some content unavailable, but much of the content ends up being heavily replicated.
- Scalability: As the number of users and amount of content increase, so does the number of servers; protocols do not require “all-to-all” communication or coordination.

However, up to now, there are limitations that come with these advantages. The schema and queries for searching for content are typically hardwired into the application; there can be bottlenecks at the centralized index; there are no mechanisms for combining or otherwise manipulating the content itself. Recently there is interest in adapting the P2P model to distributed data management scenarios, which try to overcome weak query capabilities and limitations in index scalability.

These current limitations are acceptable for file-sharing applications, since people find ways to encode metadata about a file in the filename, but more general P2P applications for public service use will require a richer query model. In terms of index scalability and result quality both the Napster and Gnutella approaches have serious limitations. Centralized index servers don’t scale with the number of clients. Query broadcasting wastes network bandwidth and hurts result quality by limiting the availability of rare content. Again, file-sharing networks thrive despite these limitations: Finding 10 out of the 100 available copies of the same file is

[13] cf. Papadimos V. et al., “Distributed Query Processing and Catalogs for Peer-to-Peer Systems”, http://www.cs.uoi.gr/~pitoura/courses/p2p/papers/papadimos03.pdf

usually good enough, but not for general purpose P2P query systems used in a public service environment.

Peers express their preferences for the data they are serving or looking for using a name space of multiple hierarchical categories. Queries are routed efficiently, without depending on centralized index servers or query broadcasting, and peers can make intelligent choices about query latency, data completeness and currency tradeoffs. Here, the important distinction between the P2P model and the client-server model is not that classical roles do not exist, but that they are not fixed or pre-assigned; this query's client may well become the next query's server.

Given the considerable traffic volume a P2P application, it is crucial that it employs well available networking resources. The scalability of a P2P application is ultimately determined by how efficiently it uses the underlying network. For example, Gnutella's store-and-forward architecture makes the virtual network topology extremely important. The larger the mismatch between the network infrastructure and the P2P application's virtual topology, the bigger the "stress" on the infrastructure.

P2P systems have tended to focus on the integration of simple resources (individual computers) via protocols designed to provide specific vertically integrated functionality. Thus, for example, Gnutella defines its own protocols for search and network maintenance. Such protocols do, of course, define an infrastructure, but in general (at least for second and third generation systems) the persistence properties of such infrastructures are not specifically engineered. Over time, experience with these properties has revealed the need for new services, such as reliability management, anonymity and reputation management.

9 References

[DHS06]	“SECURITY IN THE SOFTWARE LIFECYCLE”, U.S. Department of Homeland Security, 2006, Chapter 4.5.2.2 Life Cycle Phase-Specific Practices on p. 82
[LL06]	Lientz, Larrsen, “Risk Management for IT Projects”, 2006, p. xv
[RED06]	Redwine, “Software Assurance”, US Departments of Homeland Security and Defense, 2006, Chapter 6.6. Software Reuse on p.99
[D2.1]	State of the Art Report. Deliverable D2.1, Access-eGov Project, 2006.
[D2.2]	User Requirement Analysis & Development/Test Recommendations. Deliverable D2.2, Access-eGov Project, 2006.